305-CD-009-001

EOSDIS Core System Project

Release A SDPS Ingest Subsystem Design Specification

July 1995

Hughes Information Technology Corporation
Landover, MD

Release A SDPS Ingest Subsystem Design Specification for the ECS Project

July 1995

Prepared Under Contract NAS5-60000
Item #046

SUBMITTED BY

Parag Ambardekar /s/	7/28/95
Parag Ambardekar, Release A CCB Chairman	Date
EOSDIS Core System Project	

Hughes Information Technology Corporation
Landover, Maryland

Preface

This document is one of sixteen comprising the detailed design specifications of the SDPS and CSMS subsystem for Release A of the ECS project. A complete list of the design specification documents is given below. Of particular interest are documents number 305-CD-004, which provides an overview of the subsystems and 305-CD-018, the Data Dictionary, for those reviewing the object models in detail. A Release A SDPS and CSMS CDR Review Guide (510-TP-002) is also available.

The SDPS and CSMS subsystem design specification documents for Release A of the ECS Project include:

305-CD-004	Release A Overview of the SDPS and CSMS Segment System Design Specification
305-CD-005	Release A SDPS Client Subsystem Design Specification
305-CD-006	Release A SDPS Interoperability Subsystem Design Specification
305-CD-007	Release A SDPS Data Management Subsystem Design Specification
305-CD-008	Release A SDPS Data Server Subsystem Design Specification
305-CD-009	Release A SDPS Ingest Subsystem Design Specification
305-CD-010	Release A SDPS Planning Subsystem Design Specification
305-CD-011	Release A SDPS Data Processing Subsystem Design Specification
305-CD-012	Release A CSMS Segment Communications Subsystem DesignSpecification
305-CD-013	Release A CSMS Segment Systems Management Subsystem Design Specification
305-CD-014	Release A GSFC Distributed Active Archive Center Implementation
305-CD-015	Release A LaRC Distributed Active Archive Center Implementation
305-CD-016	Release A MSFC Distributed Active Archive Center Implementation
305-CD-017	Release A EROS Data Center Distributed Active Archive Center Implementation
305-CD-018	Release A Data Dictionary for Subsystem Design Specification
305-CD-019	Release A System Monitoring and Coordination Center Implementation

Object models presented in this document have been exported directly from CASE tools and in some cases contain too much detail to be easily readable within hard copy page constraints. The reader is encouraged to view these drawings on line using the Portable Document Format (PDF) electronic copy available via the ECS Data Handling System (ECS) at URL http://edhs1.gsfc.nasa.gov.

This document is a contract deliverable with an approval code 2. As such, it does not require formal Government approval, however, the Government reserves the right to request changes within 45 days of the initial submittal. Once approved, contractor changes to this document are handled in accordance with Class I and Class II change control requirements described in the EOS Configuration Management Plan, and changes to this document shall be made by document change notice (DCN) or by complete revision.

Any questions should be addressed to:

Data Management Office
The ECS Project Office
Hughes Information Technology Corporation
1616 McCormick Drive
Landover, MD 20785

Abstract

The Ingest Subsystem consists of a collection of hardware and software that supports the ingest of data into ECS repositories. This volume presents the overview and critical design of the Ingest CSCI and Ingest Client HWCI elements that comprise this subsystem.

Keywords: Ingest, PDL, CSCI, HWCI, client host, working storage, network ingest, polling ingest, hard media ingest, preprocessing, session, request, metadata, GUI, Level 0

This page intentionally left blank.

Change Information Page

List of Effective Pages	
Page Number	**Issue**
Title	Final
iii through xv	Final
1-1 and 1-2	Final
2-1 through 2-4	Final
3-1 through 3-14	Final
4-1 through 4-180	Final
5-1 through 5-12	Final
A-1 through A-12	Final
B-1 through B-16	Final
AB-1 through AB-4	Final
GL-1 through GL-8	Final

Document History			
Document Number	**Status/Issue**	**Publication Date**	**CCR Number**
305-CD-009-001	Final	July 1995	95-0472

This page intentionally left blank.

Contents

5. ICLHW - Ingest Client HWCI

Figures

This page intentionally left blank.

1. Introduction

1.1 Identification

This Release A SDPS Ingest Subsystem Design Specification for the ECS Project, Contract Data Requirement List (CDRL) Item 046, with requirements specified in Data Item Description (DID) 305/DV2, is a required deliverable under the Earth Observing System Data and Information System (EOSDIS) Core System (ECS), Contract NAS5-60000. This publication is part of a series of documents comprising the Science and Communications Development Office design specification for the Communications and System Management Segment (CSMS) and the Science and Data Processing Segment (SDPS) for Release A.

1.2 Scope

The Release A SDPS Ingest Subsystem Design Specification defines the detailed design of the Ingest Subsystem. It defines the Ingest Subsystem computer software and hardware architectural design, as well as subsystem design based on Level 4 requirements.

This subsystem is on a formal development track. It is released in and reviewed at the formal Release A Critical Design Review.

This document reflects the June 21, 1995 Technical Baseline maintained by the contractor configuration control board in accordance with ECS Technical Direction No.11, dated December 6, 1994.

1.3 Document Organization

The document is organized to describe the Release A SDPS Ingest Subsystem design as follows:

Section 1 provides information regarding the identification, scope, status, and organization of this document.

Section 2 provides a listing of the related documents, which were used as source information for this document.

Section 3 provides an overview of the Subsystem, focusing on the high-level design concept. This provides general background information to put Ingest into context.

Section 4 contains the structure of the computer software configuration item (CSCI) comprising the Ingest Subsystem. Included are CSCI context diagrams, the CSCI object model, the CSCI dynamic model (scenarios), and the CSCI physical model (executables).

Section 5 contains the hardware configuration item (HWCI) design of the Ingest Subsystem.

Appendix A provides the Level 4 requirements-to-design mapping matrix for use in verifying coverage of the Level 4 requirements.

Appendix B contains the Program Design Language (PDL) for all non-trivial Ingest object model operations.

An Acronym list and Glossary are provided.

1.4 Status and Schedule

This submittal of DID 305/DV2 meets the milestone specified in the CDRL of NASA contract NAS5-60000. The submittal was reviewed during the SDPS Preliminary Design Review (PDR) and reflects changes to the design which resulted from that review. The PDR also triggered a number of follow up actions in response to Review Item Discrepancies (RID) the results of which are incorporated into the Critical Design Review (CDR) version of this document.

2. Related Documents

2.1 Parent Documents

The parent document is the document from which the scope and content of this Ingest Subsystem Design Specification is derived.

194-207-SE1-001	System Design Specification for the ECS Project
305-CD-002-002	Science and Data Processing Segment (SDPS) Design Specification for the ECS Project

2.2 Applicable Documents

The following documents are referenced within this Subsystem Design Specification, or are directly applicable, or contain policies or other directive matters that are binding upon the content of this volume.

209-CD-001-001	Interface Control Document Between EOSDIS Core System (ECS) and the NASA Science Internet
209-CD-002-001	Interface Control Document Between EOSDIS Core System (ECS) and ASTER Ground Data System
209-CD-003-001	Interface Control Document Between EOSDIS Core System (ECS) and EOS-AM Project for AM-1 Spacecraft Analysis Software
209-CD-004-001	Data Format Control Document for the Earth Observing System (EOS) AM-1 Project Data Base
209-CD-005-002	Interface Control Document Between EOSDIS Core System (ECS) and Science Computing Facilities (SCF)
209-CD-006-002	Interface Control Document Between EOSDIS Core System (ECS) and National Oceanic and Atmospheric Administration (NOAA) Affiliated Data Center (ADC)
209-CD-007-002	Interface Control Document Between EOSDIS Core System (ECS) and TRMM Science Data and Information System (TSDIS)
209-CD-008-002	Interface Control Document Between EOSDIS Core System (ECS) and the Goddard Space Flight Center (GSFC) Distributed Active Archive Center (DAAC)
209-CD-009-002	Interface Control Document Between EOSDIS Core System (ECS) and the Marshall Space Flight Center (MSFC) Distributed Active Archive Center (DAAC)
209-CD-011-002	Interface Control Document Between EOSDIS Core System (ECS) and the Version 0 System
305-CD-003-002	Communications and System Management Segment (CSMS) Design Specification for the ECS Project

308-CD-001-004	Software Development Plan for the ECS Project
313-CD-004-001	Release A Communications and System Management Segment (CSMS) and Science Data Processing Segment (SDPS) Internal Interface Control Document for the ECS Project
423-41-03	Goddard Space Flight Center, EOSDIS Core System (ECS) Contract Data Requirements Document

2.3 Information Documents Not Referenced

The following documents, although not referenced herein and/or not directly applicable, do amplify and clarify the information presented in this document. These documents are not binding on the content of the Subsystem Design Specifications.

205-CD-002-002	Science User's Guide and Operations Procedure Handbook for the ECS Project. Part 4: Software Developer's Guide to Preparation, Delivery, Integration, and Test with ECS
206-CD-001-002	Version 0 Analysis Report for the ECS Project
209-CD-010-001	Interface Control Document Between EOSDIS Core System (ECS) and the Langley Research Center (LaRC) Distributed Active Archive Center (DAAC)
194-302-DV2-001	ECS Facilities Plan for the ECS Project
101-303-DV1-001	Individual Facility Requirements for the ECS Project, Preliminary
194-317-DV1-001	Prototyping and Studies Plan for the ECS Project
318-CD-003-XXX	Prototyping and Studies Progress Report for the ECS Project (monthly)
333-CD-003-001	SDP Toolkit Users Guide for the ECS Project
601-CD-001-003	Maintenance and Operations Management Plan for the ECS Project
194-604-OP1-001	ECS Operations Concept Document for the ECS Project, Working Draft
101-620-OP2-001	List of Recommended Maintenance Equipment for the ECS Project
194-703-PP1-001	System Design Review (SDR) Presentation Package for the ECS Project
193-801-SD4-001	PGS Toolkit Requirements Specification for the ECS Project
194-813-SI4-002	Planning and Scheduling Prototype Results Report for the ECS Project
194-813-SI4-003	DADS Prototype One FSMS Product Operational Evaluation
194-813-SI4-004	DADS Prototype One STK Wolfcreek 9360 Automated Cartridge System Hardware Characterization Report
813-RD-009-001	DADS Prototype Two Multi-FSMS Product Integration Evaluation
828-RD-001-002	Government Furnished Property for the ECS Project
193-WP-118-001	Algorithm Integration and Test Issues for the ECS Project
193-WP-611-001	Science-based System Architecture Drivers for the ECS Project, Revision 1.0
193-WP-623-001	ECS Evolutionary Development White Paper

194-WP-901-002	EOSDIS Core System Science Information Architecture, White Paper, Working Paper
194-WP-902-002	ECS Science Requirements Summary, White Paper, Working Paper
194-WP-913-003	User Environment Definition for the ECS Project, White Paper, Working Paper
194-WP-925-001	Science Software Integration and Test, White Paper, Working Paper
420-WP-001-001	Maximizing the Use of COTS Software in the SDPS SDS Software Design, White Paper
160-TP-002-001	Data Migration White Paper
194-TP-548-001	User Scenario Functional Analysis [for the ECS Project]
194-TP-569-001	PDPS Prototyping at ECS Science and Technology Laboratory, Progress Report #4
222-TP-003-005	Release Plan Content Description for the ECS Project
430-TP-001-001	SDP Toolkit Implementation with Pathfinder SSM/I Precipitation Rate Algorithm, Technical Paper
440-TP-001-001	Science Data Server Architecture Study [for the ECS Project]
440-TP-014-001	ECS Ingest Subsystem Topology Analysis
none	Hughes Training, Inc., ECS User Interface Style Guide, White Paper, Version 5.0
423-16-01	Goddard Space Flight Center, Data Production Software and Science Computing Facility (SCF) Standards and Guidelines
423-41-02	Goddard Space Flight Center, Functional and Performance Requirements Specification for the Earth Observing System Data and Information System (EOSDIS) Core System
540-022	Goddard Space Flight Center, Earth Observing System (EOS) Communications (Ecom) System Design Specification
560-EDOS-0211.0001	Goddard Space Flight Center, Interface Requirements Document Between EDOS and the EOS Ground System (EGS)

This page intentionally left blank.

3. Ingest Subsystem Overview

3.1 Introduction and Context

The Release A SDPS Ingest Subsystem contains a collection of hardware and software that supports the ingest of data into ECS repositories on a routine and ad hoc basis and triggers subsequent archiving and/or processing of the data. The Ingest Subsystem configuration must be flexible to support a variety of data formats and structures, external interfaces, and ad-hoc ingest tasks. Data processing and storage functions to be performed by the Ingest Subsystem and ingest clients vary according to attributes of the ingested data such as data type, data format, and the level to which the ingested data has been processed.

From a software perspective, the Ingest Subsystem is organized into a collection of tools from which those required for a specific situation can be configured. The resultant configuration is called an ingest client. Ingest clients may exist in a static configuration to service a routine external interface, or they may be specially configured and exist only for the duration of a specific ad hoc ingest task. The ingest clients provide a single virtual interface point for the receipt of all external data to be archived within the SDPS. Individual ingest clients are established to support each unique interface, allowing the interface parameters to be modified as interface and mission requirements evolve. Ingest data preprocessing, metadata validation, and metadata extraction is performed by the ingest clients on any incoming data, as required.

Data is staged to one of two areas depending on the data level, data type, and other data set specific characteristics:

- Level 0 data from ongoing missions will be staged to the Ingest Subsystem working storage area, where the data will be ingested and stored in the Level 0 (L0) rolling store. The staged data will also be accessible by the SDPS Processing Subsystem for that data which must be processed to higher levels.
- Level 1a-4 data will be staged directly to the working storage area in the Data Server Subsystem. Ingest client functionality such as quality checking and reading of metadata will be performed on this data upon the Data Server Subsystem processor hardware. The data server will then archive the data in the logical and physical data server to which the particular data has been assigned.

The hardware components of the Ingest Subsystem are similar to those of the data server, but are specialized to meet the ingest requirements at a given site. Specialized forms of ingest clients may be incorporated into site unique architectures, and additional processing hardware may also be incorporated at those sites where special transformations must be accomplished on ingest data sets.

Ingest Subsystem hardware presented in this section consists of the Ingest Client HWCI. Since the Ingest Subsystem is a form of data server, it utilizes hardware which is similar to the data server Access Control and Management HWCI, Working Storage HWCI, and Data Repository HWCI. The relationship of the Ingest Client HWCI with these other HWCIs is discussed in Section 5.

3.1.1 Ingest Subsystem Context Diagram

The Ingest Subsystem must be capable of accepting data from a variety of sources including both electronic network interfaces and hard media. Early interface testing is performed at Interim Release-1 (IR-1) for interfaces at the Sensor Data Processing Facility (SDPF), the Tropical Rainfall Measuring Mission (TRMM) Science Data Information System (TSDIS), and the NOAA Affiliated Data Centers (ADCs). The NOAA ADCs include the National Environmental Satellite, Data, and Information Service (NESDIS) and the GSFC Data Assimilation Office (GDAO). Release A interfaces include the SDPF, TSDIS, NOAA ADCs (NESDIS and the GDAO), the Science Computing Facilities (SCFs) (for algorithm delivery), the Data Server Subsystem (for archiving), science users, clients (operations staff), Version 0 DAACs, and other ECS DAACs. Early interface testing is performed at Release A for the EOS Data and Operations System (EDOS), Landsat-7, and EOS Operations Center (EOC) interfaces. Additional interfaces are added at Release B and at future releases. The Ingest Subsystem context diagram is shown in Figure 3.1-1. Specific details on the interfaces are included in Table 3.1-1.

The following assumptions have been made regarding the characteristics of the data to be ingested:

- EDOS and SDPF will each generate production data sets which will be transferred when the data set is complete. EDOS has stated that Production Data Sets (PDSs) will based on time and/or size (not granule or orbit). Each transferred file would contain only one Application Process ID (APID). SDPF has stated that the data will be transferred once per 24 hour period.
- Receipt and processing of products from other sources is assumed to be random, but largely continuous over a 24-hour period. Ingest Subsystem resources will be sized to handle predictable peak loads.

3.2 Ingest Subsystem Overview

3.2.1 Ingest Subsystem Configuration Item (CI) List

The Ingest Subsystem is composed of one Computer System Configuration Item (CSCI) and one Hardware Configuration Item (HWCI):

- Ingest CSCI
- Ingest Client HWCI

These CIs are discussed in detail in subsections 4 and 5, respectively.

3.2.2 Ingest Subsystem Design Rationale

Main drivers for the design are:

- the high reliability required for Level 0 data ingest
- the required extensibility of the ingest client implementation to future external interfaces
- the demands which are imposed on ingest by the migration of Version 0 data from DAAC repositories external to ECS; and
- other performance requirements related to data ingest

The Ingest Subsystem design incorporates the following measures in response to the above drivers:

- The need for high reliability to support the function of Level 0 science data ingest was resolved by the logical and physical separation of the Ingest Subsystem (Level 0) data server instantiation from other data server instantiations. The ingest of Level 0 data has a very high priority, and must be supported with high component reliability and availability. Maintaining this level of reliability, maintainability, and availability (RMA) throughout the entire SDPS would be prohibitively expensive. Separating a high RMA ingest complement of hardware and software from other SDPS functions allows each subsystem within SDPS to support only the level of RMA necessary to perform its required functions.
- The need for future extensibility was resolved by separating the ingest processing component from the associated data server component and providing template interface software that may be reused as new interfaces are added or old interfaces modified. The external interfaces to be supported by the ingest clients change over time as spacecraft and instruments are added and removed. Each external interface must potentially be supported with a different data transfer mechanism, format conversion, quality checking, metadata definition, and other attributes unique to that data. Separating the performance of these functions from the Level 0 data repository component minimizes or eliminates changes to the data server configuration as mission requirements change.

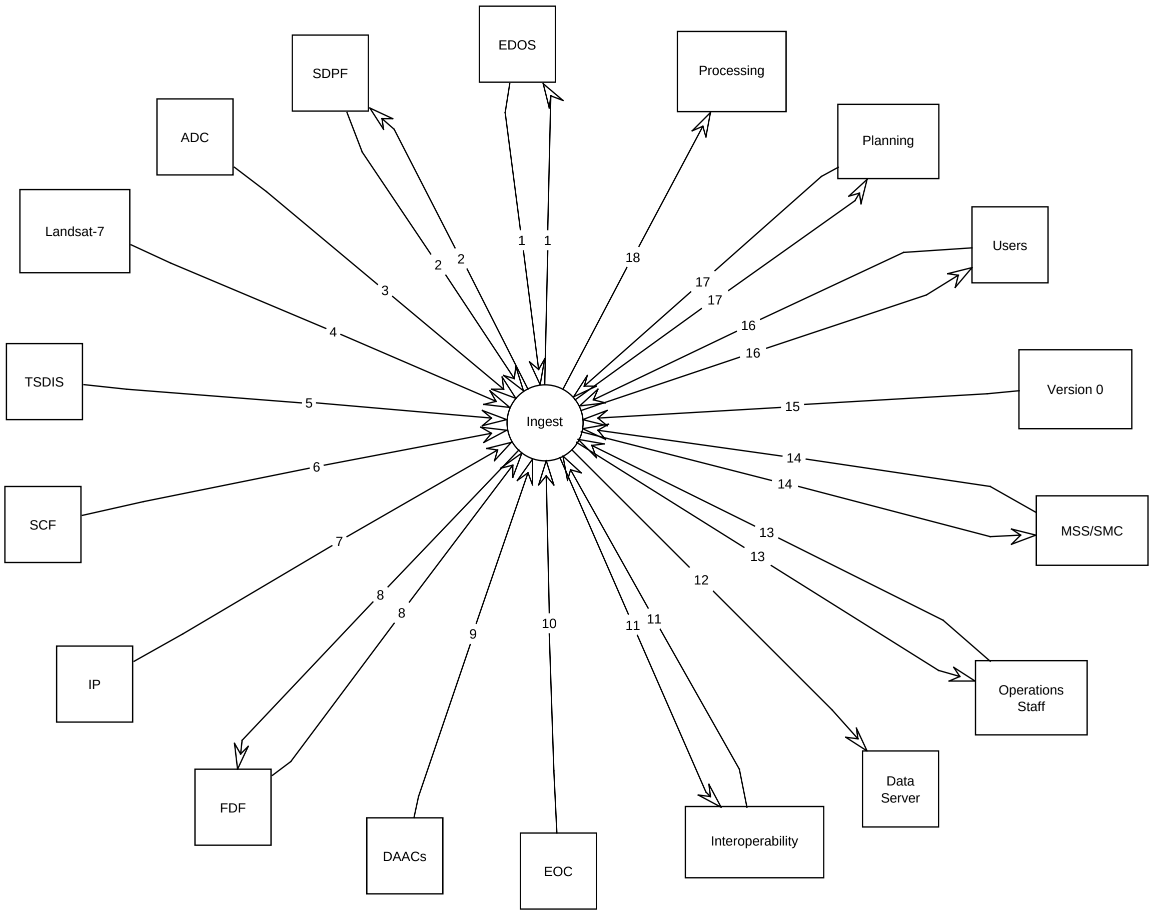


Figure 3.1-1. Ingest Subsystem Context Diagram

Table 3.1-1. Ingest Subsystem Interfaces (1 of 5)

Flow No.	Source	Destination	Data Types	Data Volume	Frequency
13	Ingest	Operations Staff	Ingest Status	less than 1 MB	in response to request
13	Ingest	Operations Staff	Ingest Threshold Update Status	less than 1 MB	in response to request
13	Ingest	Operations Staff	Ingest Log	less than 1 MB	in response to request
12	Ingest	Data Server	Standard Products	greater than 1 GB	as required for archiving
12	Ingest	Data Server	Metadata	between 1 MB and 1 GB	as required for archiving
12	Ingest	Data Server	Ancillary Data	greater than 1 GB	as required for archiving
12	Ingest	Data Server	Correlative Data	greater than 1 GB	as required for archiving
12	Ingest	Data Server	Calibration Data	between 1 MB and 1 GB	as required for archiving
12	Ingest	Data Server	Documents	between 1 MB and 1 GB	as required for archiving
12	Ingest	Data Server	Orbit/Attitude Data	between 1 MB and 1 GB	as required for archiving
12	Ingest	Data Server	Data Availability Schedules	between 1 MB and 1 GB	as required for archiving
12	Ingest	Data Server	Algorithms	greater than 1 GB	as required for archiving
12	Ingest	Data Server	Special Products	greater than 1 GB	as required for archiving
12	Ingest	Data Server	L0 Data	greater than 1 GB	as required for archiving
12	Ingest	Data Server	Quick Look Data	between 1 MB and 1 GB	as required for archiving
12	Ingest	Data Server	QA Data	between 1 MB and 1 GB	as required for archiving
12	Ingest	Data Server	Resource Allocation / Deallocation Re-quests	less than 1 MB	on reception of data
1*	Ingest	EDOS	Service Requests (Back-up data requests)	less than 1 MB	rare
1	Ingest	EDOS	Fault Report	less than 1 MB	rare
1	Ingest	EDOS	Fault Isolation Re-quest	less than 1 MB	depends on EDOS
1	Ingest	EDOS	L0 data	greater than 1 GB	rare
8*	Ingest	FDF	Repaired/Retained Orbit Data Request	less than 1 MB	infrequent (depends on mission requirements)
11	Ingest	Interopera-bility	Search Request	less than 1 MB	as required for archiving
11	Ingest	Interopera-bility	Advertisement	less than 1 MB	when capability changes
11*	Ingest	Interopera-bility	Subscription	less than 1 MB	in response to request
14	Ingest	MSS/SMC	Ingest status	less than 1 MB	in response to request

Table 3.1-1. Ingest Subsystem Interfaces (2 of 5)

Flow No.	Source	Destination	Data Types	Data Volume	Frequency
14	Ingest	MSS/SMC	Ingest log	less than 1 MB	in response to request
17	Ingest	Planning	Notice	less than 1 MB	several times per day as L0 data arrives from SDPF or EDOS
18	Ingest	Processing	L0 data	greater than 1 GB	as requested by Processing for L0 to higher level processing
2	Ingest	SDPF	L0 data	between 1 MB and 1 GB	as requested by Processing for L0 reprocessing
16	Ingest	Users	Ingest Status	less than 1 MB	as requested
3	ADC (NESDIS and GDAO)	Ingest	Metadata	less than 1 MB	frequency dependent on user input
3	ADC (NESDIS and GDAO)	Ingest	Ancillary Data	between 1 MB and 1 GB	frequency dependent on data set
3	ADC (NESDIS and GDAO)	Ingest	Calibration Data, Correlative Data, Documents	less than 1 MB	frequency dependent on user input
3	ADC (NESDIS and GDAO)	Ingest	Data Availability Notices	less than 1 MB	several times a day
13	Client Operations Staff	Ingest	ingest status requests	less than 1 MB	as requested
13	Operations Staff	Ingest	ingest log requests	less than 1 MB	as requested
13	Operations Staff	Ingest	ingest control requests	less than 1 MB	as requested
13	Operations Staff	Ingest	ingest threshold control requests	less than 1 MB	as requested
9	DAACs	Ingest	Ancillary Data	greater than 1 GB	as required
9	DAACs	Ingest	Correlative Data	greater than 1 GB	as required
9	DAACs	Ingest	Calibration Data	between 1 MB and 1 GB	as required
9	DAACs	Ingest	QA Data	between 1 MB and 1 GB	as required
1	EDOS	Ingest	Service Request Disposition	less than 1 MB	as required

Table 3.1-1. Ingest Subsystem Interfaces (3 of 5)

Flow No.	Source	Destination	Data Types	Data Volume	Frequency
1	EDOS	Ingest	PDSs (L0 Data)	greater than 1 GB	several times a day
1*	EDOS	Ingest	ADSs (Back-up L0 Data)	greater than 1 GB	as required
1	EDOS	Ingest	PDS Delivery Record	less than 1 MB	several times a day
1*	EDOS	Ingest	ADS Delivery Record	less than 1 MB	as required
1*	EDOS	Ingest	Physical Media Unit Delivery Record	less than 1 MB	as required
1	EDOS	Ingest	Undetected Fault Iso-lation	less than 1 MB	as required
10	EOC	Ingest	Data Availability No-tice	less than 1 MB	several times a day
10	EOC	Ingest	Telemetry Data	173 MB/day	Twice per day (12 files each transfer)
10	EOC	Ingest	Event	12 MB/day	Every hour
10	EOC	Ingest	EOC statistics, sched-ules, reports, etc.	25 MB/day	Twice per day (25 files total each day)
8*	FDF	Ingest	Repaired Orbit Data	between 1 MB and 1 GB	as requested
11	Interop-erability	Ingest	Notification	very low	as requested
11	Interop-erability	Ingest	Advertisement Info[a]	less than 1 MB	in response to request
7*	IP	Ingest	Project Data Base In-formation	less than 1 MB	as required (every 3 months)
7*	IP	Ingest	History Data	between 1 MB and 1 GB	daily (Media Delivery from ASTER)
7*	IP	Ingest	Level 0 - Level 4 Prod-ucts	greater than 1 GB	daily (For ASTER, media delivery & electronic for Q/L Level 1a)
7*	IP	Ingest	Metadata	less than 1 MB	dependent on user input
7*	IP	Ingest	Schedule Adjudica-tion Data	less than 1 MB	as required
7*	IP	Ingest	Status	less than 1 MB	as required
7*	IP	Ingest	Documents	less than 1 MB	as required
7*	IP	Ingest	Calibration Data	between 1 MB and 1 GB	as required
7*	IP	Ingest	Correlative Data	between 1 MB and 1 GB	as required
7*	IP	Ingest	Ancillary Data	between 1 MB and 1 GB	as required
4	Land-sat 7 PS	Ingest	Metadata	less than 1 MB	several times a day
4	Land-sat 7 PS	Ingest	L0 Science Data	~140 GB/day	several times a day

Table 3.1-1. Ingest Subsystem Interfaces (4 of 5)

Flow No.	Source	Destination	Data Types	Data Volume	Frequency
4	Land-sat 7 MOC	Ingest	Activity Calendar	between 1 MB and 1 GB	TBR
4	Land-sat 7 PS	Ingest	Payload Correction Data	between 1 MB and 1 GB	several times a day
4	Land-sat 7 PS	Ingest	Mirror Scan Correc-tion Data	between 1 MB and 1 GB	several times a day
4	Land-sat 7 PS	Ingest	Calibration data	between 1 MB and 1 GB	several times a day
4	Land-sat 7 PS	Ingest	Data Availability No-tice	less than 1 MB	several times a day
4	Land-sat 7 PS	Ingest	Browse Data	greater than 1 GB	as required
4	Land-sat 7 PS	Ingest	Directory and Guide Information	between 1 MB and 1 GB	TBR
4	Land-sat 7 IGS	Ingest	Inventory Data	between 1 MB and 1 GB	TBR
4	Land-sat 7 IGS	Ingest	Browse Data	greater than 1 GB	TBR
4	Land-sat 7 IAS	Ingest	Calibration Data	between 1 MB and 1 GB	TBR
4	Land-sat 7 IAS	Ingest	Metadata	between 1 MB and 1 GB	TBR
14	MSS/ SMC	Ingest	Ingest status requests	less than 1 MB	in response to request
14	MSS/ SMC	Ingest	Ingest log requests	less than 1 MB	in response to request
17	Plan-ning	Ingest	Subscription	less than 1 MB	as required
6*	SCF	Ingest	Status	less than 1 MB	as required
6*	SCF	Ingest	Metadata/updates	less than 1 MB	as required
6*	SCF	Ingest	Documents	less than 1 MB	as required
6	SCF	Ingest	Algorithms/Updates	between 1 MB and 1 GB	as required
2	SDPF	Ingest	L0 Data	65 MB/day - MSFC 90 MB/day - LaRC	daily
2	SDPF	Ingest	Quick Look Data	between 1 MB and 1 GB	three times a day
2	SDPF	Ingest	Predictive Orbit Data	between 1 MB and 1 GB	daily
2	SDPF	Ingest	Definitive Orbit Data	between 1 MB and 1 GB	daily

Table 3.1-1. Ingest Subsystem Interfaces (5 of 5)

Flow No.	Source	Destination	Data Types	Data Volume	Frequency
2	SDPF	Ingest	Data Availability Notice	less than 1 MB	daily
2	SDPF	Ingest	Back-up Data	between 1 MB and 1 GB	as required
5	TSDIS	Ingest	Metadata	1 days worth of products	daily
5	TSDIS	Ingest	Data Availability Notice	less than 1 MB	daily
5	TSDIS	Ingest	Data Products	60 (GB/day) Processing and Reprocessing	
5	TSDIS	Ingest	Algorithms	between 1 MB and 1 GB	as required
5	TSDIS	Ingest	Documents	less than 1 MB	as required
5	TSDIS	Ingest	Status	less than 1 MB	as required
5	TSDIS	Ingest	Browse Data	149 MB/day	daily
5	TSDIS	Ingest	Directory	less than 1 MB	as required
5	TSDIS	Ingest	Guide	less than 1 MB	as required
5	TSDIS	Ingest	Schedule Adjudication Data	less than 1 MB	as required
16	Users	Ingest	User Methods	between 1 MB and 1 GB	as required
16	Users	Ingest	Ingest Status Requests	less than 1 MB	as required
15	Version 0	Ingest	Migration Data	greater than 1 GB	varies depending on migration strategy

a. Items marked with an asterisk (*) in the Flow No. column are interfaces to be implemented post-Release A.

b. Ingest uses the Advertisement Information to locate the relevant Data Servers with which it needs to interact.

c. In the table, where an exact number is unavailable, the data volume is estimated as low (less than 1 MB), medium (between 1 MB and 1 GB), or high (greater than 1 GB) per use defined in the frequency column The frequency information will be updated as the interfaces are fully defined. Note that EDOS, EOC, and Landsat-7 interfaces are implemented only to the extent needed for purposes of early interface testing at Release A.

- In addition, each new or modified external interface may require custom interface software to facilitate the data transfer process. The long-term EOS program expects to add large numbers of new interfaces over time. The Ingest Subsystem software is designed in a modular fashion so as to minimize the development effort required for new or modified interfaces.
- The volume and complexity of data provided from the Version 0 facilities to the ECS for archival are critical design drivers for the ECS Ingest Subsystem. Over 600 data products have been identified, ranging in volume from megabytes to hundreds of gigabytes. Total volume is on the order of dozens of terabytes. Many of the data products are stored in some

form of Hierarchical Data Format (HDF); however, many more products are stored in other formats. The Ingest Subsystem software is designed to generalize the mechanism by which data is routinely stored within the SDPS, given a set of standalone tools used to prepare Version 0 data. Version 0 "data preparation" includes retrieval from Version 0-specific hard media, conversion to EOS-HDF, where required, and extraction of standard metadata.

- ECS satisfies explicit performance requirements with the design described in Sections 4 and 5 of this volume.

The following subsections elaborate upon the above design drivers.

3.2.2.1 Ingest RMA Architecture

A principal objective of separating ingest from other SDPS functions is to assure the high reliability and availability of the system for Level 0 data ingest. Ingest availability requirements are met through the use of high reliability components in redundant configurations, as necessary. The following paragraphs provide additional detail on ingest RMA requirements and how the ingest architecture ensures that these requirements will be met.

The principal Level 0 data sources (e.g., EDOS, SDPF) each support a data driven architecture that processes data within 24 hours or less from receipt of the data from the spacecraft. Once the data is processed into Production Data Sets (PDSs) (with size based on clock time, number of packets, or Tracking and Data Relay Satellite System [TDRSS] service session) the data must be transferred to ECS in a timely fashion for archiving and any required higher level processing. Typically, data transfer must be completed within several hours to free up resources at the Level 0 processing sites, as new data sets are being received on a nearly continuous basis. The Level 0 processing sites provide long term archiving of the PDSs in the event that data is corrupted or lost in the transfer to ECS or within ECS itself. While it is possible to access these archived data sets if they are needed, it is unattractive to do so from an operational standpoint due to the added time and complexity of pulling the data from the deep archive and staging it for transfer. Therefore, the SDPS function of receiving science data availability requirement of 0.999 must be met to reduce the number of required data retransmissions and ensure that the needs of the overall data system are met. The following section describes how this requirement and others which place RMA requirements on the Ingest Subsystem of 2000 hours mean time between failures (MTBF) and 15 minutes mean time to restore (MTTR) from a failure to operational capability will be met. A more detailed availability analysis may be found in the Availability Models/Predictions for the ECS Project document (515-CD-001-002).

It should be noted that certain of the sites at each ECS release are not staffed 24 hours per day, and are subject to somewhat different operational requirements during unstaffed periods. The Ingest Subsystem is designed to operate with minimal operator involvement, and is planned to continue the function of ingesting data during unstaffed periods. However, certain functions that inherently require initial operator involvement, such as initiating ingest via hard media, will not be supported during unstaffed periods. Moreover, faults may require human involvement to reconfigure and reboot the ingest client hosts. Therefore, the 15 minute MTTR requirement for support of the receipt of science data will not be supported during unstaffed periods. In order to eliminate any potential system overload, Ingest Subsystem capabilities are being sized to satisfy full 24 hour L0 data ingest requirements during staffed periods.

The instantiation of the Ingest Subsystem varies at each site, but is based on the same architecture concepts and classes of hardware and software. The generic Ingest Subsystem architecture is shown in Figure 3.1-2. The ingest client software required for a specific Level 0 interface at a given site runs on a client host computer residing in the Ingest Subsystem. Multiple ingest clients may run on a single client host, or may be divided among multiple client hosts, depending on the data load supported by each interface. At least one spare client host is provided at each site in order to provide a warm backup failover capability in the event of a primary host failure. A hot backup approach was initially considered, but was determined to be unnecessary and undesirable for several reasons. Current EDOS requirements and architecture concepts do not support the transmission of data to multiple targets with different application identifiers (APIDs). This makes the implementation of a prime and hot spare ingest client host configuration more complicated, but a warm spare configuration where the backup server may be quickly reconfigured to replace the prime unit was determined to be sufficient to meet requirements. Commercially available workstations of the class required for the client hosts typically support MTBFs in the 20,000 to 40,000 hour range. Calculations indicate that the use of this hardware in conjunction with redundant working storage and data repository components will meet data ingest RMA requirements.

Working storage and Level 0 data repository media drive and robotics devices are sized to accommodate redundant devices or components as necessary. A combination of paper analysis and system modeling efforts have been used to determine the final configuration necessary to meet system performance and RMA requirements. A summary of the Ingest Subsystem sizing analysis is presented in the ECS Ingest Subsystem Topology Analysis (440-TP-014-001).

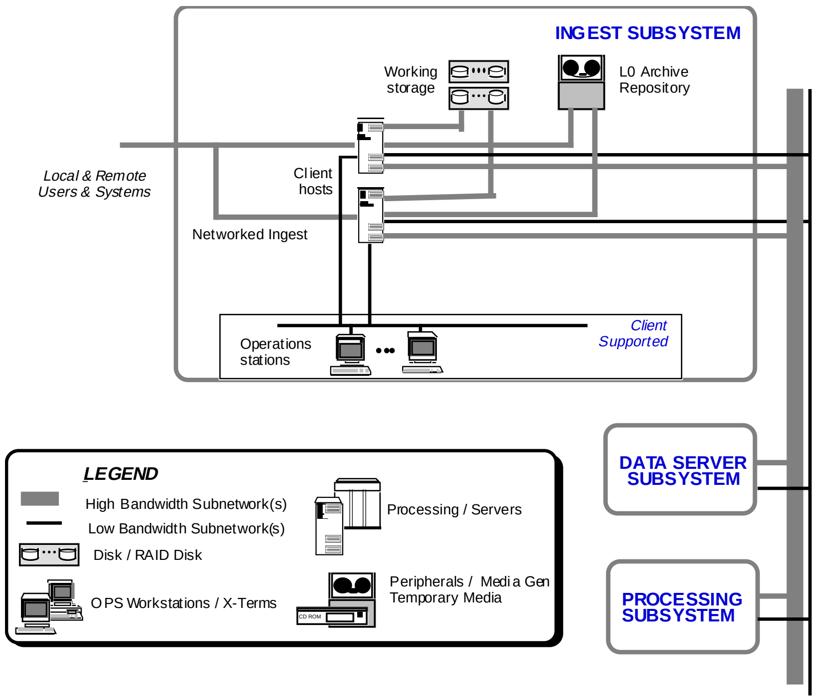


Figure 3.1-2. Ingest Subsystem Hardware Diagram

3.2.2.2 Ingest Client Implementation

As shown in the context diagram in subsection 3.1.1, the Ingest Subsystem supports a wide variety of external interfaces. The application-level protocol to set up for data transfer is potentially different for each of the external interfaces. As a result, a separate ingest client software application is required to facilitate data transfer for each interface. To minimize the software development effort and make it easier to accommodate new external interfaces in the future, the external interfaces were categorized based on common characteristics as follows:

- Automated Network Ingest by means of a Data Availability Notice (DAN) supplied to ECS--ECS receives the DAN and schedules automated network data transfer from the source. The DAN describes the location of the available data. ECS "gets" data from the source within a specified time window. Note: External data providers are responsible for developing application software to interact with ECS automated network ingest software. The SDPF has existing design/software that may be used as a template.
- Polling Ingest with Delivery Record--ECS periodically checks an agreed-upon network location for a Delivery Record file. The Delivery Record file contains information identical to that in a DAN. The Delivery Record describes the location of the available data. Note: the data location may be on a working storage device within ECS, where an external data provider may have previously transferred the data. If a Delivery Record is located, ECS "gets" data from the source within a specified time window.
- Polling Ingest without Delivery Record--ECS periodically checks an agreed-upon network location for available data. All data in the location is assumed to make up a collection of ingest data of one specific data type, with one file per data granule. If data is located, ECS "gets" data from the source within a system-tunable time window.
- Manual data transfer mechanisms are in place for transfer of data from hard media and for science user-controlled (interactive) network data transfer.
  - Hard Media Ingest is available for authorized institutions or science users providing data on hard media and as a backup mechanism for facilities where automated network data transfer is temporarily unavailable. The hard media must contain information identical to the Delivery Records described above, in a standard file format, or the data provider must separately provide Delivery Records in the standard file format.
  - Interactive Network Ingest is available for authorized science users to manually identify data to be ingested. Science users may "put" the data into an accessible ECS location or may request that ECS "get" the data from their workstation. Information identical to that contained in the Delivery Record is entered by means of GUI input (or derived by the GUI software).

Data transfer is accomplished by one of three means--file transfer protocol (ftp) "get", ftp "put", or hard media data transfer. Ftp get involves ECS "getting" data from an external site. Ftp put involves an external site "putting" data into ECS. Hard media data transfer involves data transfer from one of several ingest peripheral types found at a DAAC. Kerberized ftp (kftp), providing additional communications security services, is specified as the standard data ingest protocol. Exceptions may be made in the case of interfaces to existing facilities, which may be granted waivers from supporting kftp on a case-by-case basis.

Table 3.1-2 describes each external interface in Table 3.1-1 in terms of the interface protocols used, and the ICD in which detailed interface design information may be found:

Table 3.1-2. External Interface Protocols (1 of 2)

Interface (facility)	Type of Primary Interface Protocols	Type of Backup Interface Protocols	Comments
SDPF	Automated Network Ingest/ftp get	8mm tape	Defined by SDPF ICD
TSDIS	Automated Network Ingest/ftp get	8mm tape	Defined by ECS/TSDIS ICD
NOAA/ NMC (DAO)	Polling Ingest without Delivery Record/ftp get	None	Defined by ECS/GSFC DAAC-specific ICD
NOAA (NESDIS)	Polling Ingest without Delivery Record/ftp get	None	Defined by ECS/ ADC(NOAA) ICD
EDOS	Polling Ingest with Delivery Record/ftp put	TBD hard media	Defined by EDOS ICD (TBS)
ASTER DPS*	TBD hard media	None	Defined by ECS/IP ICD (TBS)
EOC	Automated Network Ingest/ftp get	None	Defined by internal interface agreement
FDF*	TBD	TBD	Defined by ECS/FDF ICD (TBS)
Landsat-7	Automated Network Ingest/ftp get	None	Defined by Landsat-7 ICD (TBS)
SCF	Automated Network Ingest/ftp get	8mm tape	Defined by ECS/SCF ICD
Users*	Interactive Network Ingest/ftp get or put	None	Defined by TBD
Version 0 (GSFC)	Hard Media Ingest (8mm tape) and/or Automated Network Ingest/ftp get and/or Interactive Network Ingest/ftp get or put	None	Defined by ECS/GSFC DAAC-specific ICD
Version 0 (LaRC)	Automated Network Ingest/ftp get plus Interactive Network Ingest/ftp get or put	TBD	Defined by ECS/LaRC DAAC-specific ICD
Version 0 (MSFC)	Hard Media Ingest (8mm tape)	None	Defined by ECS/MSFC DAAC-specific ICD
Version 0 (JPL)*	TBD	TBD	Defined by the TBS ECS/ JPL DAAC-specific ICD
Version 0 (NSIDC)*	TBD	TBD	Defined by the TBS ECS/ NSIDC DAAC-specific ICD
Version 0 (ASF)*	TBD	TBD	Defined by the TBS ECS/ ASF DAAC-specific ICD

Table 3.1-2. External Interface Protocols (2 of 2)

Interface (facility)	Type of Primary Interface Protocols	Type of Backup Interface Protocols	Comments
Version 0 (EDC)*	TBD	TBD	Defined by the TBS ECS/ EDC DAAC-specific ICD
Version 0 (CIESIN)*	TBD	TBD	Defined by the TBS ECS/ CIESIN DAAC-specific ICD
Version 0 (ORNL)*	TBD	TBD	Defined by the TBS ECS/ ORNL DAAC-specific ICD

*Interfaces marked with an asterisk are implemented post-Release A; note that EDOS, EOC, and Landsat-7 interfaces are implemented only to the extent needed for purposes of early interface testing at Release A.

3.2.2.3 Version 0 Migration Impact on the Ingest Subsystem

Requirements that dictate the volume and other characteristics of Version 0 data to be migrated in the Release A timeframe from the Version 0 DAACs into the ECS are a critical design driver. The Version 0 data widely varies in format, structure, volume, and transfer mechanism. Previous design team experience indicates that migration of existing data requires a major engineering effort, including data analysis; data conversion and reformatting tool development; extensive integration and test (to ensure data integrity); system analysis of required hardware components (e.g., networks, storage, etc.); and maintenance and operations (to perform the actual migration and data validation).

The design team has proposed a process to ensure the successful migration of Version 0 data into the ECS. The process is documented in a separate Data Migration Plan white paper (160-TP-002-001, Version 1). That white paper describes the process by which Version 0 data is transformed into a standard form that is recognizable by the Ingest Subsystem. The white paper assumes that the Ingest Subsystem is structured to facilitate ingest of data in a standard form.

As described in the following sections, the Ingest Subsystem is structured so that data is ingested into ECS by a standard mechanism once the data is preprocessed (e.g., data conversions and reformatting, metadata extraction, and metadata quality checking). Therefore, the primary task of Version 0 migration--data preprocessing--may be performed externally from the Ingest Subsystem. In addition, the Ingest Subsystem design provides standard ingest interfaces (as described in previous subsections). Therefore, the selection of the data transfer mechanism (e.g., 8mm tape, network transfer) for a specific Version 0 data product may require additional copies of Ingest Subsystem hardware, but does not require new (i.e., undesigned) Ingest Subsystem hardware or software. The potential use of a data transfer media that is not already supported in Release A would need to be evaluated on a case-by-case basis.

Accordingly, the Version 0 migration effort may be planned as a separate activity from the Ingest Subsystem development. A proposal for development of a "Version 0 Migration Facility" has been formulated and will be reported on at the Critical Design Review (CDR).

3.2.2.4 Impact of Performance Requirements

TRMM Expedited Data Ingest

The Ingest CSCI assigns different priorities to each external site (e.g., the SDPF). The capability to assign a different priority to each category of data within the site (e.g., production Level 0 data/ expedited (equivalent to quicklook) data) is available as an extension to the design, but is not currently implemented. Our analysis indicates that the rate of ingest of SDPF data (delivery of Level 0 data once per day; delivery of expedited data three times per day) and the relatively small data volumes (much less than one gigabyte) may be handled by the existing priority scheme.

TRMM I/O Throughput

Based upon Level 0 data volumes and loading estimated in Appendix A of the TRMM IRD ECS modeled the nominal data I/O throughput. Note: the non-Level 0 data flow into Data Server hardware, and therefore do not impact the Ingest HWCI. In addition, the following TRMM IRD requirements for TSDIS reprocessing imply the peak throughput:

TRMM3100 -- ...ECS shall also daily ingest an average of 2 days worth of reprocessed data from TSDIS. (MSFC interface)

TRMM4090 -- ...ECS shall also daily ingest an average of 2 days worth of reprocessed data from TSDIS. (GSFC interface)

Assuming that Level 0 data is retrieved for reprocessing at this same rate, the Ingest HWCI is sized for three days' data volume (one day's worth of Level 0 data ingest, two days' worth of Level 0 data retrieved from storage). The hardware sizing information is documented in section 5 of this document and in the DAAC-specific volumes for LaRC and MSFC.

4. INGST - Ingest CSCI

4.1 CSCI Overview

The Ingest CSCI is responsible for the receipt of data arriving at a site and the physical placement of data into the site's storage hierarchy. A provider site within EOSDIS will normally need to ingest a wide variety of data types to support the services it wishes to offer. These data may be delivered through different interfaces (network file transfer, hard media, hard copy, etc.), with varying management approaches to these interfaces. This interface heterogeneity and the need to support extendibility and new data/interfaces as algorithms and provider functionality changes, lead to a design in which the ingest functionality is isolated from other subsystems within the segment design.

Although each instance of the Ingest CSCI has to deal with the characteristics of the specific external interface it is managing, the general functionality is similar in each case.

- Ingest processing is either event-driven or timer-driven. For automated network ingest (e.g., SDPF), data centers send Data Availability Notices to the DAACs to indicate the availability of data. For hard media ingest, the "data availability notice" is entered by DAAC operations staff at a GUI interface. Similarly, for interactive network ingest under science user control, the "data availability notice" is entered by the science user at a GUI interface. For timer-driven ingest (e.g., NESDIS), on the other hand, data centers transfer data to an agreed-upon network location and ECS ingest clients periodically check ("poll") for the existence of new data.
- Depending on the interface, data may be transferred by either a data "get" or a data "put". A data get is performed by the Ingest CSCI under Ingest CSCI control. A data put is performed by another data center under that data center's control.
- The Ingest CSCI performs transmission checks relevant to the transfer mechanism (e.g. data quality, data gaps, data redundancy, missing files, etc.) and notifies the data source of success or failure. Failure results in a request to resend or in notification of the operations staff. The DAAC operations staff monitors the status of active ingest processing.
- The Ingest CSCI extracts sufficient metadata to allow the data to be retrieved at a later time from the Data Server. The metadata information is contained within the data file, within a separate metadata file in standard format associated with the data, or within the information provided to request ingest. The form in which metadata is provided is determined by the Interface Control Document (ICD) defined for the specific I/F. Some portion of the metadata is checked for quality (e.g., all required metadata parameters available, parameters within a range of values, etc.). Additional metadata, such as the time of ingest, is determined by the Ingest CSCI.
- When the collection of data is complete (i.e., all referenced data are available), the Ingest CSCI requests insertion of the data into an appropriate Data Server.

- The Ingest CSCI records the successful or unsuccessful transfer of data into the site in an ingest history log, with detailed information in an error log. The DAAC operations staff and System Management Center (SMC) staff may interrogate the ingest history log and the error log. In addition, the Ingest CSCI returns the completion status for the ingest data transfer to the data ingest requester.

4.2 CSCI Context

The context diagram for the Ingest CSCI is identical to that of the Ingest Subsystem, since the Ingest CSCI is the only CSCI for the subsystem (see Figure 3.1-1). Table 4.2-1 shows the CSCI service interfaces provided to entities external to the CSCI.

Table 4.2-1. Ingest CSCI Service Interfaces

Interface	Input Data	Output Data	Description
Ingest Server.Create Session	External Data Provider	Ingest_Status	Sets up an ingest request session connection .
Ingest Session. Receive Message	Data Availability Notice Data Delivery Ack	Data Availability Ack Data Delivery Notice	Provides the application-level protocol for automated network data transfer.
Media Ingest Session. Receive Message	Hard Media Ingest Request	Ingest_Status	Provides authorized operations staff the means to enter, by means of GUI input, information required to ingest hard media.
Network Ingest Session. Receive Message	Network Ingest Request	Ingest_Status	Provides authorized science users the means to enter, by means of GUI input, information required to ingest data by network data transfer.
Status Monitor. Receive Request	User_Identifier	Ingest Request (list)	Provides the status of a) all or selected ongoing ingest requests (authorized operations staff) or b) ongoing requests for a given user (specific user).
Log Monitor. Receive Message	Ingest Log Request	Ingest Log	Provides the status of all or selected completed ingest requests to authorized operations staff.

4.3 Ingest CSCI Object Model

Figures 4.3-1 through 4.3-5 show the object classes that model the Ingest Subsystem. Subsequent paragraphs describe each of the object classes in terms of their parent class, purpose and description, and their critical attributes, operations, and associations. Note: Program Design Language (PDL) for all non-trivial object operations is included in Appendix B of this document.

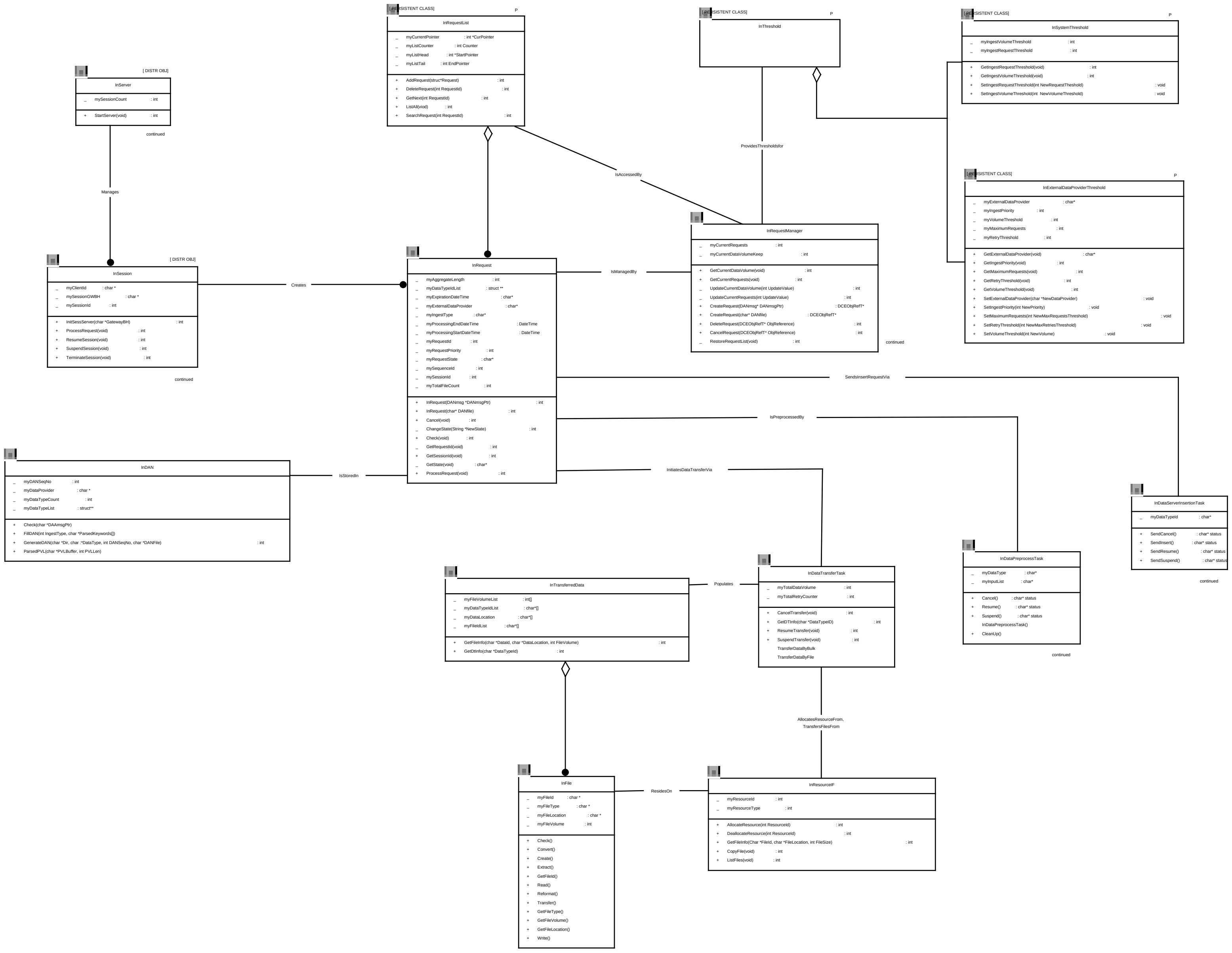


Figure 4.3-1. In_Ingest_Main_Object_Model Diagram

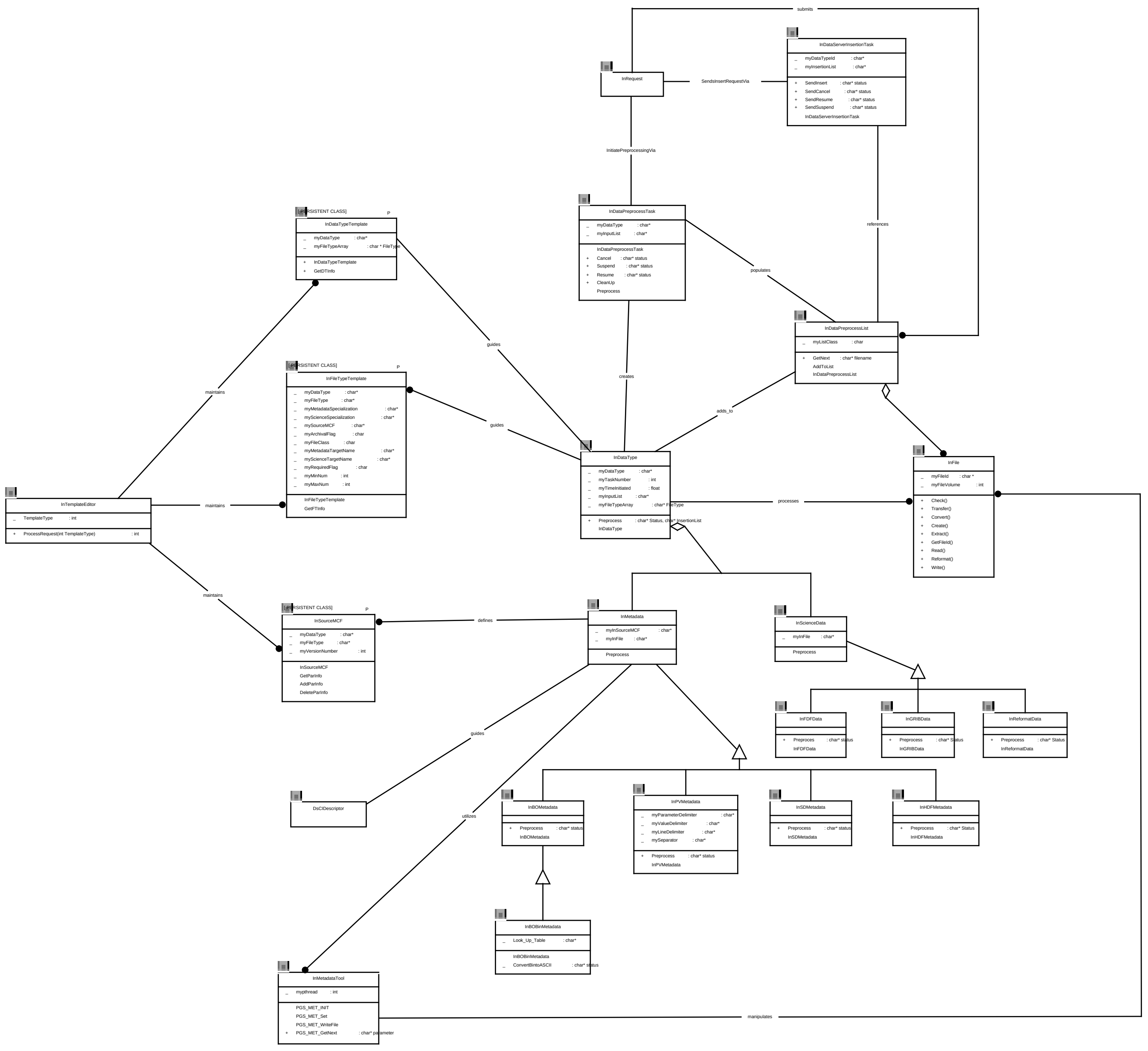
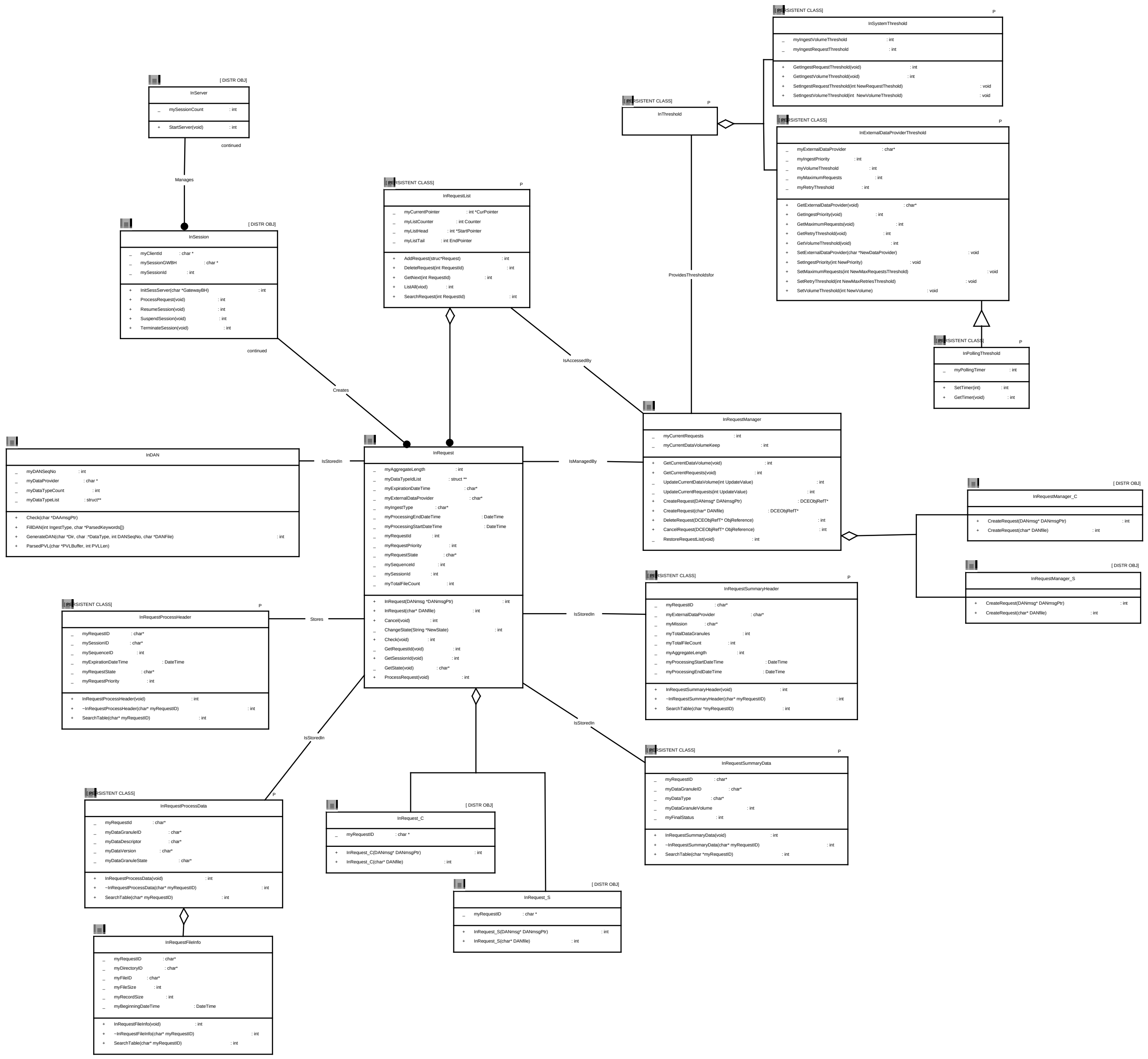
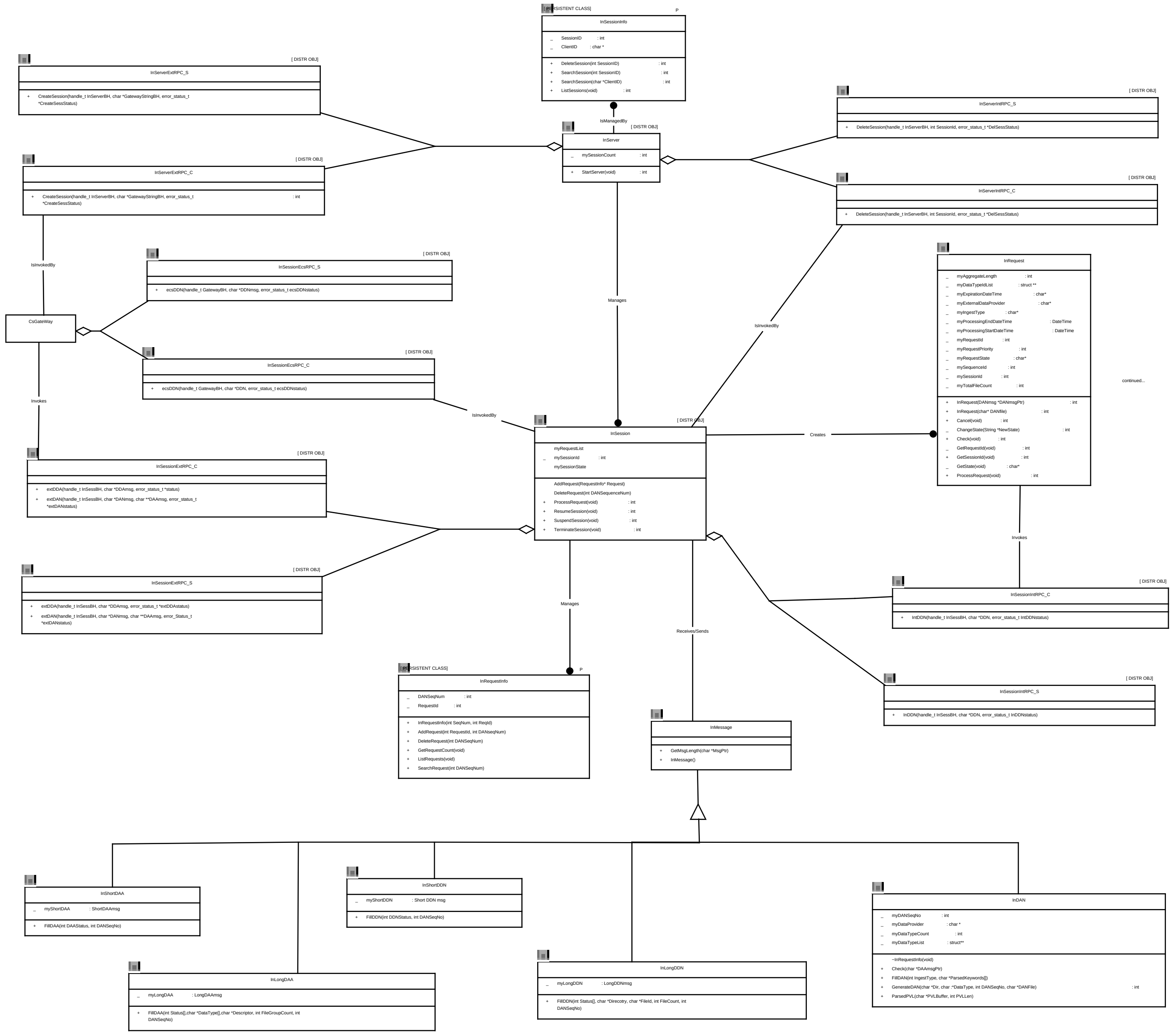


Figure 4.3-2. In_Ingest_Preprocessing Object Model Diagram,



[DISTR OBJ]
InServer
_ mySessionCount : int
+ StartServer(void) : int
continued
Manages
[DISTR OBJ]
InSession
_ myClientId : char *
_ mySessionGWBH : char *
_ mySessionId : int
+ InitSessServer(char *GatewayBH) : int
+ ProcessRequest(void) : int
+ ResumeSession(void) : int
+ SuspendSession(void) : int
+ TerminateSession(void) : int
continued
Creates
[PERSISTENT CLASS]
P
InRequestList
_ myCurrentPointer : int *CurPointer
_ myListCounter : int Counter
_ myListHead : int *StartPointer
_ myListTail : int EndPointer
+ AddRequest(struc*Request) : int
+ DeleteRequest(int RequestId) : int
+ GetNext(int RequestId) : int
+ ListAll(viod) : int
+ SearchRequest(int RequestId) : int
IsAccessedBy
[PERSISTENT CLASS]
P
InThreshold
ProvidesThresholdsfor
[PERSISTENT CLASS]
P
InSystemThreshold
_ myIngestVolumeThreshold : int
_ myIngestRequestThreshold : int
+ GetIngestRequestThreshold(void) : int
+ GetIngestVolumeThreshold(void) : int
+ SetIngestRequestThreshold(int NewRequestTheshold) : void
+ SetIngestVolumeThreshold(int NewVolumeThreshold) : void
[PERSISTENT CLASS]
P
InExternalDataProviderThreshold
_ myExternalDataProvider : char*
_ myIngestPriority : int
_ myVolumeThreshold : int
_ myMaximumRequests : int
_ myRetryThreshold : int
+ GetExternalDataProvider(void) : char*
+ GetIngestPriority(void) : int
+ GetMaximumRequests(void) : int
+ GetRetryThreshold(void) : int
+ GetVolumeThreshold(void) : int
+ SetExternalDataProvider(char *NewDataProvider) : void
+ SetIngestPriority(int NewPriority) : void
+ SetMaximumRequests(int NewMaxRequestsThreshold) : void
+ SetRetryThreshold(int NewMaxRetriesThreshold) : void
+ SetVolumeThreshold(int NewVolume) : void
[PERSISTENT CLASS]
P
InPollingThreshold
_ myPollingTimer : int
+ SetTimer(int) : int
+ GetTimer(void) : int
InRequestManager
_ myCurrentRequests : int
_ myCurrentDataVolumeKeep : int
+ GetCurrentDataVolume(void) : int
+ GetCurrentRequests(void) : int
_ UpdateCurrentDataVolume(int UpdateValue) : int
_ UpdateCurrentRequests(int UpdateValue) : int
+ CreateRequest(DANmsg* DANmsgPtr) : DCEObjRefT*
+ CreateRequest(char* DANfile) : DCEObjRefT*
+ DeleteRequest(DCEObjRefT* ObjReference) : int
+ CancelRequest(DCEObjRefT* ObjReference) : int
_ RestoreRequestList(void) : int
[DISTR OBJ]
InRequestManager_C
+ CreateRequest(DANmsg* DANmsgPtr) : int
+ CreateRequest(char* DANfile) : int
[DISTR OBJ]
InRequestManager_S
+ CreateRequest(DANmsg* DANmsgPtr) : int
+ CreateRequest(char* DANfile) : int
InDAN
_ myDANSeqNo : int
_ myDataProvider : char *
_ myDataTypeCount : int
_ myDataTypeList : struct**
+ Check(char *DAAmsgPtr)
+ FillDAN(int IngestType, char *ParsedKeywords[])
+ GenerateDAN(char *Dir, char :*DataType, int DANSeqNo, char *DANFile) : int
+ ParsedPVL(char *PVLBuffer, int PVLLen)
IsStoredIn
InRequest
_ myAggregateLength : int
_ myDataTypeIdList : struct **
_ myExpirationDateTime : char*
_ myExternalDataProvider : char*
_ myIngestType : char*
_ myProcessingEndDateTime : DateTime
_ myProcessingStartDateTime : DateTime
_ myRequestId : int
_ myRequestPriority : int
_ myRequestState : char*
_ mySequenceId : int
_ mySessionId : int
_ myTotalFileCount : int
+ InRequest(DANmsg *DANmsgPtr) : int
+ InRequest(char* DANfile) : int
+ Cancel(void) : int
_ ChangeState(String *NewState) : int
+ Check(void) : int
_ GetRequestId(void) : int
+ GetSessionId(void) : int
_ GetState(void) : char*
+ ProcessRequest(void) : int
IsManagedBy
IsStoredIn
[PERSISTENT CLASS]
P
InRequestSummaryHeader
_ myRequestID : char*
_ myExternalDataProvider : char*
_ myMission : char*
_ myTotalDataGranules : int
_ myTotalFileCount : int
_ myAggregateLength : int
_ myProcessingStartDateTime : DateTime
_ myProcessingEndDateTime : DateTime
+ InRequestSummaryHeader(void) : int
+ ~InRequestSummaryHeader(char* myRequestID) : int
+ SearchTable(char *myRequestID) : int
[PERSISTENT CLASS]
P
InRequestProcessHeader
_ myRequestID : char*
_ mySessionID : char*
_ mySequenceID : int
_ myExpirationDateTime : DateTime
_ myRequestState : char*
_ myRequestPriority : int
+ InRequestProcessHeader(void) : int
+ ~InRequestProcessHeader(char* myRequestID) : int
+ SearchTable(char* myRequestID) : int
Stores
IsStoredIn
IsStoredIn
[PERSISTENT CLASS]
P
InRequestSummaryData
_ myRequestID : char*
_ myDataGranuleID : char*
_ myDataType : char*
_ myDataGranuleVolume : int
_ myFinalStatus : int
+ InRequestSummaryData(void) : int
+ ~InRequestSummaryData(char* myRequestID) : int
+ SearchTable(char *myRequestID) : int
[PERSISTENT CLASS]
P
InRequestProcessData
_ myRequestId : char*
_ myDataGranuleID : char*
_ myDataDescriptor : char*
_ myDataVersion : char*
_ myDataGranuleState : char*
+ InRequestProcessData(void) : int
+ ~InRequestProcessData(char* myRequestID) : int
+ SearchTable(char* myRequestID) : int
[DISTR OBJ]
InRequest_C
_ myRequestID : char *
+ InRequest_C(DANmsg* DANmsgPtr) : int
+ InRequest_C(char* DANfile) : int
[DISTR OBJ]
InRequest_S
_ myRequestID : char *
+ InRequest_S(DANmsg* DANmsgPtr) : int
+ InRequest_S(char* DANfile) : int
InRequestFileInfo
_ myRequestID : char*
_ myDirectoryID : char*
_ myFileID : char*
_ myFileSize : int
_ myRecordSize : int
_ myBeginningDateTime : DateTime
+ InRequestFileInfo(void) : int
+ ~InRequestFileInfo(char* myRequestID) : int
+ SearchTable(char* myRequestID) : int

Figure 4.3-3. In_Ingest_Request_Processing_Object_Model Diagram



[PERSISTENT CLASS]
InSessionInfo
SessionID : int
ClientID : char *
DeleteSession(int SessionID) : int
SearchSession(int SessionID) : int
SearchSession(char *ClientID) : int
ListSessions(void) : int
IsManagedBy
[DISTR OBJ]
InServer
mySessionCount : int
StartServer(void) : int
InServerExtRPC_S
CreateSession(handle_t InServerBH, char *GatewayStringBH, error_status_t *CreateSessStatus)
InServerExtRPC_C
CreateSession(handle_t InServerBH, char *GatewayStringBH, error_status_t *CreateSessStatus) : int
InServerIntRPC_S
DeleteSession(handle_t InServerBH, int SessionId, error_status_t *DelSessStatus)
InServerIntRPC_C
DeleteSession(handle_t InServerBH, int SessionId, error_status_t *DelSessStatus)
IsInvokedBy
CsGateWay
InSessionEcsRPC_S
ecsDDN(handle_t GatewayBH, char *DDNmsg, error_status_t ecsDDNstatus)
InSessionEcsRPC_C
ecsDDN(handle_t GatewayBH, char *DDN, error_status_t ecsDDNstatus)
Manages
IsInvokedBy
InRequest
myAggregateLength : int
myDataTypeIdList : struct **
myExpirationDateTime : char*
myExternalDataProvider : char*
myIngestType : char*
myProcessingEndDateTime : DateTime
myProcessingStartDateTime : DateTime
myRequestId : int
myRequestPriority : int
myRequestState : char*
mySequenceId : int
mySessionId : int
myTotalFileCount : int
InRequest(DANmsg *DANmsgPtr) : int
InRequest(char* DANfile) : int
Cancel(void) : int
ChangeState(String *NewState) : int
Check(void) : int
GetRequestId(void) : int
GetSessionId(void) : int
GetState(void) : char*
ProcessRequest(void) : int
continued...
Invokes
InSession
myRequestList
mySessionId : int
mySessionState
AddRequest(RequestInfo* Request)
DeleteRequest(int DANSequenceNum)
ProcessRequest(void) : int
ResumeSession(void) : int
SuspendSession(void) : int
TerminateSession(void) : int
Creates
InSessionExtRPC_C
extDDA(handle_t InSessBH, char *DDAmsg, error_status_t *status)
extDAN(handle_t InSessBH, char *DANmsg, char **DAAmsg, error_status_t *extDANstatus)
InSessionExtRPC_S
extDDA(handle_t InSessBH, char *DDAmsg, error_status_t *extDDAstatus)
extDAN(handle_t InSessBH, char *DANmsg, char **DAAmsg, error_Status_t *extDANstatus)
InSessionIntRPC_C
IntDDN(handle_t InSessBH, char *DDN, error_status_t IntDDNstatus)
InSessionIntRPC_S
InDDN(handle_t InSessBH, char *DDN, error_status_t InDDNstatus)
Manages
Receives/Sends
InRequestInfo
DANSeqNum : int
RequestId : int
InRequestInfo(int SeqNum, int ReqId)
AddRequest(int RequestId, int DANSeqNum)
DeleteRequest(int DANSeqNum)
GetRequestCount(void)
ListRequests(void)
SearchRequest(int DANSeqNum)
InMessage
GetMsgLength(char *MsgPtr)
InMessage()
InShortDAA
myShortDAA : ShortDAAmsg
FillDAA(int DAAStatus, int DANSeqNo)
InShortDDN
myShortDDN : Short DDN msg
FillDDN(int DDNStatus, int DANSeqNo)
InLongDAA
myLongDAA : LongDAAmsg
FillDAA(int Status[],char *DataType[],char *Descriptor, int FileGroupCount, int DANSeqNo)
InLongDDN
myLongDDN : LongDDNmsg
FillDDN(int Status[], char *Direcotry, char *FileId, int FileCount, int DANSeqNo)
InDAN
myDANSeqNo : int
myDataProvider : char *
myDataTypeCount : int
myDataTypeList : struct**
~InRequestInfo(void)
Check(char *DAAmsgPtr)
FillDAN(int IngestType, char *ParsedKeywords[])
GenerateDAN(char *Dir, char *DataType, int DANSeqNo, char *DANFile) : int
ParsedPVL(char *PVLBuffer, int PVLLen)

Figure 4.3-4. In_Ingest_Session_Manager_Object Diagram

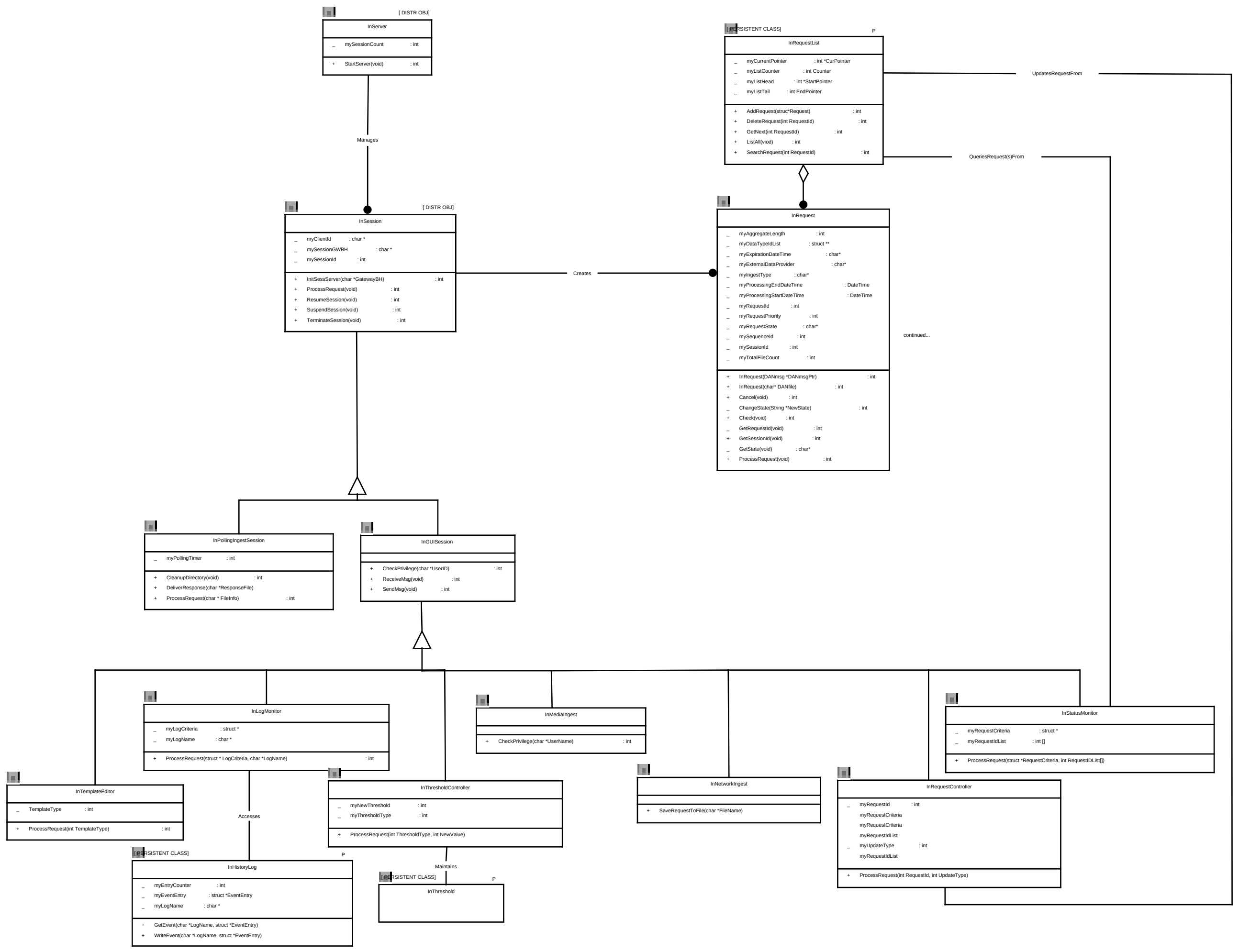

[DISTR OBJ]
InServer
_ mySessionCount : int
+ StartServer(void) : int
Manages
[DISTR OBJ]
InSession
_ myClientId : char *
_ mySessionGWBH : char *
_ mySessionId : int
+ InitSessServer(char *GatewayBH) : int
+ ProcessRequest(void) : int
+ ResumeSession(void) : int
+ SuspendSession(void) : int
+ TerminateSession(void) : int
Creates
[PERSISTENT CLASS]
P
InRequestList
_ myCurrentPointer : int *CurPointer
_ myListCounter : int Counter
_ myListHead : int *StartPointer
_ myListTail : int EndPointer
+ AddRequest(struc*Request) : int
+ DeleteRequest(int RequestId) : int
+ GetNext(int RequestId) : int
+ ListAll(viod) : int
+ SearchRequest(int RequestId) : int
UpdatesRequestFrom
QueriesRequest(s)From
InRequest
_ myAggregateLength : int
_ myDataTypeIdList : struct **
_ myExpirationDateTime : char*
_ myExternalDataProvider : char*
_ myIngestType : char*
_ myProcessingEndDateTime : DateTime
_ myProcessingStartDateTime : DateTime
_ myRequestId : int
_ myRequestPriority : int
_ myRequestState : char*
_ mySequenceId : int
_ mySessionId : int
_ myTotalFileCount : int
+ InRequest(DANmsg *DANmsgPtr) : int
+ InRequest(char* DANfile) : int
+ Cancel(void) : int
_ ChangeState(String *NewState) : int
+ Check(void) : int
_ GetRequestId(void) : int
+ GetSessionId(void) : int
_ GetState(void) : char*
+ ProcessRequest(void) : int
continued...
InPollingIngestSession
_ myPollingTimer : int
+ CleanupDirectory(void) : int
+ DeliverResponse(char *ResponseFile)
+ ProcessRequest(char * FileInfo) : int
InGUISession
+ CheckPrivilege(char *UserID) : int
+ ReceiveMsg(void) : int
+ SendMsg(void) : int
InLogMonitor
_ myLogCriteria : struct *
_ myLogName : char *
+ ProcessRequest(struct * LogCriteria, char *LogName) : int
InMediaIngest
+ CheckPrivilege(char *UserName) : int
InStatusMonitor
_ myRequestCriteria : struct *
_ myRequestIdList : int []
+ ProcessRequest(struct *RequestCriteria, int RequestIDList[])
InTemplateEditor
_ TemplateType : int
+ ProcessRequest(int TemplateType) : int
InThresholdController
_ myNewThreshold : int
_ myThresholdType : int
+ ProcessRequest(int ThresholdType, int NewValue)
InNetworkIngest
+ SaveRequestToFile(char *FileName)
InRequestController
_ myRequestId : int
myRequestCriteria
myRequestCriteria
myRequestIdList
_ myUpdateType : int
myRequestIdList
+ ProcessRequest(int RequestId, int UpdateType)
Accesses
[PERSISTENT CLASS]
P
InHistoryLog
_ myEntryCounter : int
_ myEventEntry : struct *EventEntry
_ myLogName : char *
+ GetEvent(char *LogName, struct *EventEntry)
+ WriteEvent(char *LogName, struct *EventEntry)
Maintains
[PERSISTENT CLASS]
P
InThreshold

Figure 4.3-5. In_Ingest_Session_Object_Model Diagram

4.3.1 CsGateWay Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Purpose and Description:
The Gateway translates TCP/IP socket call to the corresponding RPC function.

Attributes:

None

Operations:

None

Associations:

The CsGateWay class has associations with the following classes:
Class: InSessionExtRPC_C Invokes - The Gateway object interfaces with the InSessionExtRPC object to deliver the DAN and DDA data messages received from the external Client to Ingest Session.
Class: InServerExtRPC_C IsInvokedBy - The Gateway object interfaces with the InServerExtRPC to initiate a new Ingest Session through the Ingest Server.

4.3.2 DsClDescriptor Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Purpose and Description:
This class is a Data Server Subsystem class and therefore its attributes and operations are defined in the Data Server Subsystem documentation. The DsClDescriptor class provides the Preprocessing CSC services to access targetMCFs and validate metadata files.

Attributes:

None

Operations:

Associations:

The DsCIDescriptor class has associations with the following classes:

Class: InMetadata guides - The DsCIDescriptor class guides the InMetadata class by providing services to access target MCFs and validate metadata files.

4.3.3 InBOBinMetadata Class

Parent Class: InBOMetadata

Public: No

Distributed Object: No

Purpose and Description:

This class provides services to preprocess byte ordered binary data.

Attributes:

Look_Up_Table - This attribute defines the appropriate look-up table for conversion from binary to ASCII.

Data Type: char*

Privilege: Private

Default Value:

Operations:

ConvertBintoASCII - This operation converts binary data to ASCII values through the use of a data/file type specific look-up table.

Arguments:

Return Type: char* status

Privilege: Private

InBOBinMetadata - This is the constructor service.

Arguments:

Associations:

The InBOBinMetadata class has associations with the following classes:
None

4.3.4 InBOMetadata Class

Parent Class: InMetadata
Public: No
Distributed Object: No
Purpose and Description:
This class provides services to preprocess byte ordered data.

Attributes:

All Attributes inherited from parent class

Operations:

InBOMetadata - This is the constructor class.
Arguments:

Preprocess - This operation will extract values from byte ordered input files based on byte position information. It will access the correct MCFs and interact with the InMetadataTool Class to produce a metadata file which is acceptable to the Data Server Subystem.
Arguments:
Return Type: char* status
Privilege: Public

Associations:

The InBOMetadata class has associations with the following classes:
None

4.3.5 InDAN Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Purpose and Description:

This is the DAN (Data Availability Notice) is received from the external Client. The object class contains services to access information in the DAN.

Attributes:

myDANSeqNo - The identifier of the DAN data message.
Data Type: int
Privilege: Private
Default Value:

myDataProvider - Indicates who provides the data for ingest.
Data Type: char *
Privilege: Private
Default Value:

myDataTypeCount - Indicates the total number of data types in the DAN.
Data Type: int
Privilege: Private
Default Value:

myDataTypeList - This is a list that contains information about the file (e.g., file name, size, location).
Data Type: struct**
Privilege: Private
Default Value:

Operations:

Check - Verifies the integrity of the DAN components.
Arguments: char *DAAmsgPtr
Return Type: Void
Privilege: Public

FillDAN - Fills the DAN information into the class data memory after the DAN information is parsed.
Arguments: int IngestType, char *ParsedKeywords[]
Return Type: Void
Privilege: Public

GenerateDAN - Generates a DAN file with file information retrieved from the given directory location.
Arguments: char *Dir, char :*DataType, int DANSeqNo, char *DANFile
Return Type: int

Privilege: Public

ParsedPVL - Extracts information from the DAN and puts the information into a data memory.
Arguments: char *PVLBuffer, int PVLLen
Return Type: Void
Privilege: Public

Associations:

The InDAN class has associations with the following classes:
Class: InRequest IsStoredIn - The InDAN object information is stored in InRequest.

4.3.6 InDataPreprocessList Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Purpose and Description:
The purpose of this class is to retain lists (of files). This class provides services to add files to an existing list and retrieve files from an existing list.

Attributes:

myListClass - This attribute identifies whether the list is an input list received from the Request Processing CSC or is a list containing files to be inserted into the Data Server Subsystem.
Data Type: char
Privilege: Private
Default Value:

Operations:

AddToList - This service provides the ability to add a file to an existing list.
Arguments:

GetNext - This service provides the ability to retrieve the next file in the list.
Arguments:
Return Type: char* filename

Privilege: Public

InDataPreprocessList - This is the constructor service.
Arguments:

Associations:

The InDataPreprocessList class has associations with the following classes:
Class: InDataType adds_to - The InDataType class adds to the InDataPreprocessList class new files which have been created as a result of preprocessing.
Class: InDataPreprocessTask populates - The InDataPreprocessTask class populates the InDataPreprocessList to provide an initial list of file types to be inserted into the Data Server Subsystem.
Class: InDataServerInsertionTask references - The InDataServerInsertionTask references the InDataPreprocessList class to identify the preprocessed files which need to be inserted into the Data Server Subsystem.
Class: InRequest submits - The InRequest class submits InDataPreprocessList in order to identify the files which need to be preprocessed.

4.3.7 InDataPreprocessTask Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class:
Purpose and Description:
The main purpose of this class is to initiate and monitor required data preprocessing before insertion into the data server subsystem. The InRequest Class instantiates this class for each separate preprocessing task. The InRequest class will supply an object ID for the input file list (which contains the files associated with the preprocessing task). The InDataPreprocessTask Class instantiates the InDataType Class. This class is responsible for the control and reporting of its assigned preprocessing as directed by the InRequest Class. It is also responsible for reporting the state of a particular preprocessing task whenever the state changes. This object class also provides services to cancel, suspend, and resume preprocessing tasks.

Attributes:

myDataType - The attribute specifies the data type (e.g. metadata, science) for the Data Preprocess Task object.
Data Type: char*
Privilege: Private
Default Value:

myInputList - This attribute references the input list which contains the files to be preprocessed for the data preprocess task object.
Data Type: char*
Privilege: Private
Default Value:

Operations:

Cancel - This service provides the ability to terminate associated preprocessing.
Arguments:
Return Type: char* status
Privilege: Public

CleanUp - This service will remove all files associated with checkpoint working storage.
Arguments:
Return Type: Void
Privilege: Public

InDataPreprocessTask
Arguments:

Preprocess
Arguments:

Resume - This service reactivates the data preprocessing that has previously suspended. This service is not available in Release A.
Arguments:
Return Type: char* status
Privilege: Public

Suspend - This service provides the ability to suspend associated preprocessing. This service is not available at Release A.
Arguments:
Return Type: char* status
Privilege: Public

Associations:

The InDataPreprocessTask class has associations with the following classes:

Class: InRequest InitiatePreprocessingVia - The InRequest class initiatePreprocessingvia the InDataPreprocessTask class to start preprocessing on a specific data type and associated files.

Class: InDataType creates - The InDataPreprocessTask class creates an instance of the InDataType class for each data type granule which requires preprocessing.

Class: InDataPreprocessList populates - The InDataPreprocessTask class populates the InDataPreprocessList to provide an initial list of file types to be inserted into the Data Server Subsystem.

4.3.8 InDataServerInsertionTask Class

Parent Class: Not Applicable

Public: No

Distributed Object: No

Persistent Class:

Purpose and Description:

Responsibles for the sending of request to the appropriate Data Server based on the data type identifier. The object class has knowledge of the interface protocol with the Data Server and interfaces with the Advertising Service of the Interoperability Subsystem to determine the appropriate Data Server.

Attributes:

myDataTypeId - Data type specification that is used for determining the appropriate Data Server to which the request is sent.

Data Type: char*

Privilege: Private

Default Value:

myInsertionList - List of data files to be inserted into the Data Server.

Data Type: char*

Privilege: Private

Default Value:

Operations:

InDataServerInsertionTask

Arguments:

SendCancel - Sends insert cancellation request to the Data Server. This service interfaces with the Advertising Service CSCI of the Interoperability Subsystem to determine the appropriate Data Server.
Arguments:
Return Type: char* status
Privilege: Public

SendInsert - Sends insert request to the Data Server. The service interfaces with the Advertising Service CSCI of the Interoperability Subsystem to determine the appropriate Data Server.
Arguments:
Return Type: char* status
Privilege: Public

SendResume - Sends insert resumption request to the Data Server. The service interfaces with the Advertising Service of the Interoperability Subsystem to determine the appropriate Data Server. This service is not available at Release A.
Arguments:
Return Type: char* status
Privilege: Public

SendSuspend - Sends insert suspension request to the Data Server. The service interfaces with the Advertising Service CSCI of the Interoperability Subsystem to determine the appropriate Data Server. This service is not available in Release A.
Arguments:
Return Type: char* status
Privilege: Public

Associations:

The InDataServerInsertionTask class has associations with the following classes:

Class: InRequest SendsInsertRequestVia - The InRequest class SendsInsertRequestVia the InDataServerInsertionTask class to initiate the process necessary to insert data into the Data Server Subsystem.

Class: InDataPreprocessList references - The InDataServerInsertionTask references the InDataPreprocessList class to identify the preprocessed files which need to be inserted into the Data Server Subsystem.

4.3.9 InDataServerInsertionTask Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class:
Purpose and Description:
Responsibles for the sending of request to the appropriate Data Server based on the data type identifier. The object class has knowledge of the interface protocol with the Data Server and interfaces with the Advertising Service of the Interoperability Subsystem to determine the appropriate Data Server.

Attributes:

myDataTypeId - Data type specification that is used for determining the appropriate Data Server to which the request is sent.
Data Type: char*
Privilege: Private
Default Value:

Operations:

SendCancel - Sends insert cancellation request to the Data Server. This service interfaces with the Advertising Service CSCI of the Interoperability Subsystem to determine the appropriate Data Server.
Arguments:
Return Type: char* status
Privilege: Public

SendInsert - Sends insert request to the Data Server. The service interfaces with the Advertising Service CSCI of the Interoperability Subsystem to determine the appropriate Data Server.
Arguments:
Return Type: char* status
Privilege: Public

SendResume - Sends insert resumption request to the Data Server. The service interfaces with the Advertising Service of the Interoperability Subsystem to determine the appropriate Data Server. This service is not available at Release A.
Arguments:
Return Type: char* status
Privilege: Public

SendSuspend - Sends insert suspension request to the Data Server. The service interfaces with the Advertising Service CSCI of the Interoperability Subsystem to determine the appropriate Data Server. This service is not available in Release A.
Arguments:
Return Type: char* status
Privilege: Public

Associations:

The InDataServerInsertionTask class has associations with the following classes:
Class: InRequest SendsInsertRequestVia - The InRequest class SendsInsertRequestVia the InDataServerInsertionTask class to initiate the process necessary to insert data into the Data Server Subsystem.

4.3.10 InDataTransferTask Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class:
Purpose and Description:
It is responsible for coordinating the data transfer and the population of the InTransferredData object class. In addition, the object class is responsible for attempting the data transfer retries when data transmission failed.

Attributes:

myTotalDataVolume - The total data volume of ingest files transmitted.
Data Type: int
Privilege: Private
Default Value:

myTotalRetryCounter - The total number of data transfer retry attempts performed.
Data Type: int
Privilege: Private
Default Value:

Operations:

CancelTransfer - Cancels data transfer processing.
Arguments: void
Return Type: int
Privilege: Public

GetDTInfo - Requests for the data type grouping information for the associated ingest files.
Arguments: char *DataTypeID
Return Type: int
Privilege: Public

ResumeTransfer - Resumes the data transfer processing which has previously put to hold.
Arguments: void
Return Type: int
Privilege: Public

SuspendTransfer - Suspends the data transfer processing.
Arguments: void
Return Type: int
Privilege: Public

TransferDataByBulk
Arguments:

TransferDataByFile
Arguments:

Associations:

The InDataTransferTask class has associations with the following classes:
Class: InResourceIF AllocatesResourceFrom,TransfersFilesFrom - The InDataTransferTask object performs device allocation and bulk data transfer via the InResourceIF object.
Class: InRequest InitiatesDataTransferVia - The InRequest object interfaces with the InDataTransferTask object to perform data transfer.
Class: InTransferredData Populates - The InDataTransfer object gets information on the data types and the files from the InTransferredData object.

4.3.11 InDataType Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Purpose and Description:
This class provides services to manage the processing of files associated with a specific data type/preprocessing task. This class will contain functionality necessary to preprocess any data type. The functionality executed is driven by the InDataTypeTemplate and InFileTypeTemplate classes.

Attributes:

myDataType - This attribute specifies the data type (e.g. CER00, LIS00) for the data type object.
Data Type: char*
Privilege: Private
Default Value:

myFileTypeArray - This attribute identifies the array that stores the list of file types associated with a specific data type.
Data Type: char* FileType
Privilege: Private
Default Value:

myInputList - This attribute references the input list which contains the files to be preprocessed by the data type object.
Data Type: char*
Privilege: Private
Default Value:

myTaskNumber - This attribute specifies the task number of the data type object
Data Type: int
Privilege: Private
Default Value:

myTimeInitiated - This attribute indicates the time that data type object was created.
Data Type: float
Privilege: Private
Default Value:

Operations:

InDataType - This is the constructor service.
Arguments:

Preprocess - This service provides the ability to preprocess any data type granule. This includes controlling preprocessing of the granule, utilizing the metadata tool, and accessing target and source metadata configuration files.
Arguments:
Return Type: char* Status, char* InsertionList
Privilege: Public

Associations:

The InDataType class has associations with the following classes:
Class: InDataPreprocessList adds_to - The InDataType class adds to the InDataPreprocessList class new files which have been created as a result of preprocessing.
Class: InDataPreprocessTask creates - The InDataPreprocessTask class creates an instance of the InDataType class for each data type granule which requires preprocessing.
Class: InDataTypeTemplate guides - The InDataTypeTemplate class guides the InDataType class by providing it with the associated file types for a specific data type.
Class: InFileTypeTemplate guides - The InFileTemplate class guides the InDataType class by providing information characterizing specific file types.
Class: InFile processes - The InDataType class processes the InFile class to attain preprocessed science, ancillary, and metadata files.

4.3.12 InDataTypeTemplate Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class: True
Purpose and Description:
This class contains information that categorizes each ingest data type. The services provided enable the InDataType Class to determine what the required file types are for a given data type. The class will enable the addition of new ingest data types.

Attributes:

myDataType - This attribute specifies the data type (e.g. CER00, LIS00) for a data type template object.
Data Type: char*

Privilege: Private
Default Value:

myFileTypeArray -
Data Type: char * FileType
Privilege: Private
Default Value:

Operations:

GetDTInfo - This service provides data type specific information to be able to discern whether the appropriate files are present to do further preprocessing.
Arguments:
Return Type: Void
Privilege: Public

InDataTypeTemplate - This is the constructor service
Arguments:
Return Type: Void
Privilege: Public

Associations:

The InDataTypeTemplate class has associations with the following classes:
Class: InDataType guides - The InDataTypeTemplate class guides the InDataType class by providing it with the associated file types for a specific data type.
Class: InTemplateEditor maintains - The InTemplateEditor class maintains the InDataTypeTemplate class by providing the ability to add new instances of the InDataTypeTemplate class.

4.3.13 InExternalDataProviderThreshold Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class: True
Purpose and Description:

Persistent thresholds on an External Data Provider basis for limits on Ingest request traffic, data volumes, and data transfer retries.

Attributes:

myExternalDataProvider - Identifier of the external data provider (e.g., TSDIS) that supplies ingest requests.
Data Type: char*
Privilege: Private
Default Value:

myIngestPriority - Priority associated with ingest requests from the external data provider.
Data Type: int
Privilege: Private
Default Value:

myMaximumRequests - Maximum number of requests allowed to be procssed at one time by the external data provider.
Data Type: int
Privilege: Private
Default Value:

myRetryThreshold - Number of retries to perform when a communication failure is encountered with the external data provider.
Data Type: int
Privilege: Private
Default Value:

myVolumeThreshold - Maximum volume of data allowd across all ongoing ingest request submitted by an external data provider.
Data Type: int
Privilege: Private
Default Value:

Operations:

GetExternalDataProvider - Get the name of the external data provider
Arguments: void
Return Type: char*
Privilege: Public

GetIngestPriority - Get the ingest priority for an external data provider.

Arguments: void
Return Type: int
Privilege: Public

GetMaximumRequests - Get the maximum number of requests threshold for an external data provider.
Arguments: void
Return Type: int
Privilege: Public

GetRetryThreshold - Get the communication retries threshold for an external data provider.
Arguments: void
Return Type: int
Privilege: Public

GetVolumeThreshold - Get the data volume threshold for an external data provider.
Arguments: void
Return Type: int
Privilege: Public

SetExternalDataProvider - Set the name of the new data provider.
Arguments: char *NewDataProvider
Return Type: void
Privilege: Public

SetIngestPriority - Set the new ingest priority for an external data provider.
Arguments: int NewPriority
Return Type: void
Privilege: Public

SetMaximumRequests - Set the maxmimum ingest requests threshold for an external data provider.
Arguments: int NewMaxRequestsThreshold
Return Type: void
Privilege: Public

SetRetryThreshold - Set the communications retries threshold for an external data provider.
Arguments: int NewMaxRetriesThreshold
Return Type: void
Privilege: Public

SetVolumeThreshold - Set the data volume threshold for an external data provider.
Arguments: int NewVolume

Return Type: void
Privilege: Public

Associations:

The InExternalDataProviderThreshold class has associations with the following classes:
InThreshold (Aggregation)

4.3.14 InFDFData Class

Parent Class: InScienceData
Public: No
Distributed Object: No
Purpose and Description:
This class provides services to preprocess FDF data into acceptable data server format.

Attributes:

All Attributes inherited from parent class

Operations:

InFDFData - This is the constructor service.
Arguments:

Preproces - This operation will reformat/convert FDF data into acceptable Data Server format.
Arguments:
Return Type: char* status
Privilege: Public

Associations:

The InFDFData class has associations with the following classes:
None

4.3.15 InFile Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class:
Purpose and Description:
Instantiates an ingested file on available storage space by collaborating with the DsStResource object class services (described in the Data Server Subsystem section of this document). The InFile object class also performs the file size and file existence checks.

Attributes:

myFileId - The unique identifier of the ingest file.
Data Type: char *
Privilege: Private
Default Value:

myFileLocation - Identifies the location the file resides.
Data Type: char *
Privilege: Private
Default Value:

myFileType - Identifies the file type(e.g., metadata file, science data, calibration) of the ingest file.
Data Type: char *
Privilege: Private
Default Value:

myFileVolume - The data volume of the ingest file.
Data Type: int
Privilege: Private
Default Value:

Operations:

Check - Verifies the existence and size correctness of an ingest file.
Arguments:
Return Type: Void
Privilege: Public

Convert - Converts the ingest file to the ECS internal format.

Arguments:
Return Type: Void
Privilege: Public

Create - Creates and opens a new file.
Arguments:
Return Type: Void
Privilege: Public

Extract - Extracts data information from the ingest file.
Arguments:
Return Type: Void
Privilege: Public

GetFileId - Returns the file name of the ingest file.
Arguments:
Return Type: Void
Privilege: Public

GetFileLocation
Arguments:
Return Type: Void
Privilege: Public

GetFileType - Returns the file type (e.g., metadata file, science data, calibration data) of the ingest file.
Arguments:
Return Type: Void
Privilege: Public

GetFileVolume - Returns the size of the ingest file.
Arguments:
Return Type: Void
Privilege: Public

Read - Reads a record from the ingest file.
Arguments:
Return Type: Void
Privilege: Public

Reformat - Reformats the ingest file to the specified format.
Arguments:
Return Type: Void
Privilege: Public

Transfer - Initiates transfer of the ingest file.
Arguments:
Return Type: Void
Privilege: Public

Write
Arguments:
Return Type: Void
Privilege: Public

Associations:

The InFile class has associations with the following classes:
Class: InMetadataTool manipulates - The InMetadata class manipulates the InFile class by reading/writing files.
Class: InDataType processes - The InDataType class processes the InFile class to attain preprocessed science, ancillary, and metadata files.
InDataPreprocessList (Aggregation)

4.3.16 InFileTypeTemplate Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class: True
Purpose and Description:
This class is responsible for storing information that categorizes each ingest file type (e.g. metadata vs science data). This information is used by the InDataType Class to create the appropriate specializations of the InMetadata and InScienceData base classes. The class will contain the necessary information on how to process each specific file type.

Attributes:

myArchivalFlag - Indicates whether the file type needs to be permanently archived
Data Type: char
Privilege: Private
Default Value:

myDataType - This attribute specifies the data type (e.g. CER00, LIS00) for the file type template object.

Data Type: char*
Privilege: Private
Default Value:

myFileClass - Defines the file type further by indicating whether the file contains metadata, science data, or both.
Data Type: char
Privilege: Private
Default Value:

myFileType - This attribute specifies the file type (e.g. metadata, science) of the file type template object.
Data Type: char*
Privilege: Private
Default Value:

myMaxNum - This attributes indicates the maximum acceptable number of files for the file type object.
Data Type: int
Privilege: Private
Default Value:

myMetadataSpecialization - Specifies the correct InMetadata specialization for the specific data type
Data Type: char*
Privilege: Private
Default Value:

myMetadataTargetName - Indicates the nomenclature of the target PVL metadata file to be inserted into the data server subsystem.
Data Type: char*
Privilege: Private
Default Value:

myMinNum - This attribute indicates the minimum number of files required to be associated with the file type template object.
Data Type: int
Privilege: Private
Default Value:

myRequiredFlag - This attribute indicates whether the data associated with this file type template object is required.
Data Type: char
Privilege: Private
Default Value:

myScienceSpecialization - Specifies the correct InScienceData specialization for the specific file type
Data Type: char*
Privilege: Private
Default Value:

myScienceTargetName - Indicates the nomenclature of the preprocessed science data file
Data Type: char*
Privilege: Private
Default Value:

mySourceMCF - Specifies the correct InSourceMCF object for the each file type
Data Type: char*
Privilege: Private
Default Value:

Operations:

GetFTInfo - This service provides information to be able to properly instantiate the correct specialization of the base preprocessing classes (e.g., InMetadata, InScienceData) and the needed arguments to instantiate these specializations correctly for a given file type.
Arguments:

InFileTypeTemplate - This is the constructor service
Arguments:

Associations:

The InFileTypeTemplate class has associations with the following classes:
Class: InDataType guides - The InFileTemplate class guides the InDataType class by providing information characterizing specific file types.
Class: InTemplateEditor maintains - The InTemplateEditor Class maintains the InFileTemplate class by creating new instances of the InFileTemplate class.

4.3.17 InGRIBData Class

Parent Class: InScienceData
Public: No
Distributed Object: No
Purpose and Description:
This class will provide services to preprocess science data in GRIB format.

Attributes:

All Attributes inherited from parent class

Operations:

InGRIBData - This is the constructor service.
Arguments:

Preprocess - This operation converts data in GRIB format to HDF format.
Arguments:
Return Type: char* Status
Privilege: Public

Associations:

The InGRIBData class has associations with the following classes:
None

4.3.18 InGUISession Class

Parent Class: InSession
Public: No
Distributed Object: No
Persistent Class:
Purpose and Description:
Responsibles for reading in the operations staff/user service request interactively via the GUI screen and invokes the appropriate service to process the request. The object class is a derived object class from the InSession object class. It inherits all the data and services provided by the InSession object class.

Attributes:

All Attributes inherited from parent class

Operations:

CheckPrivilege - Verifies that the user has privilege to do the requesting service.
Arguments: char *UserID
Return Type: int
Privilege: Public

ReceiveMsg - Reads service request entered interactively by the requestor via the GUI interface.
Arguments: void
Return Type: int
Privilege: Public

SendMsg - Builds and sends request response to the requestor via the GUI interface.
Arguments: void
Return Type: int
Privilege: Public

Associations:

The InGUISession class has associations with the following classes:
None

4.3.19 InHDFMetadata Class

Parent Class: InMetadata
Public: No
Distributed Object: No
Purpose and Description:
This class will provide services to preprocess HDF metadata.

Attributes:

All Attributes inherited from parent class

Operations:

InHDFMetadata - This is the constructor service.
Arguments:

Preprocess - This operation will extract values from input files in the HDF format. This includes interfacing with the appropriate EOS-HDF classes, accessing the correct MCFs, and interacting with the InMetadataTool class to produce a metadata file which is acceptable to the Data Server Subsystem.
Arguments:
Return Type: char* Status
Privilege: Public

Associations:

The InHDFMetadata class has associations with the following classes:
None

4.3.20 InHistoryLog Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class: True
Purpose and Description:
Contains ingest history information. The object class provides services to record ingest events onto the log and to retrieve ingest events from the log.

Attributes:

myEntryCounter - Indicates the number of entries in the specified log.
Data Type: int
Privilege: Private
Default Value:

myEventEntry - Describes the event
Data Type: struct *EventEntry
Privilege: Private
Default Value:

myLogName - The name of the ingest history log.
Data Type: char *
Privilege: Private
Default Value:

Operations:

GetEvent - Retrieves ingest activity event(s) from the Ingest History Log based on the specified criteria.
Arguments: char *LogName, struct *EventEntry
Return Type: Void
Privilege: Public

WriteEvent - Records an ingest activity event into the Ingest History Log with time stamp.
Arguments: char *LogName, struct *EventEntry
Return Type: Void
Privilege: Public

Associations:

The InHistoryLog class has associations with the following classes:
Class: InLogMonitor Accesses - The InLogMonitor object interfaces with the InHistoryLog object to get the ingest history information.

4.3.21 InLogMonitor Class

Parent Class: InGUISession
Public: No
Distributed Object: No
Persistent Class:
Purpose and Description:
Provides operations peronnel the capability to monitor the Ingest History Log (an alias for Data Receipt Log) via the GUI interface. It allows operations personnel to specify the search criteria for log entries for viewing based on 1) ingest start/stop date and time, 2) data provider ID, 3) data set name, and 4) final request status. The object class is derived from the InGUISession object class. It inherits all the data and service members provided by the InGUISession.

Attributes:

myLogCriteria - The criteria information provided by operation personnel that is to be used for the Ingest History Log information searching.
Data Type: struct *
Privilege: Private
Default Value:

myLogName - The name of the log specified by operations personnel for monitoring.
Data Type: char *
Privilege: Private
Default Value:

Operations:

ProcessRequest - Invokes appropriate services to get and display the ingest history information based on the given criteria.
Arguments: struct * LogCriteria, char *LogName
Return Type: int
Privilege: Public

Associations:

The InLogMonitor class has associations with the following classes:
Class: InHistoryLog Accesses - The InLogMonitor object interfaces with the InHistoryLog object to get the ingest history information.

4.3.22 InLongDAA Class

Parent Class: InMessage
Public: No
Distributed Object: No
Purpose and Description:
This object class populates the long DAA (DAN Acknowledgement) data message to be sent to the external Client after the receipt of the DAN.

Attributes:

myLongDAA - This is the DAN Acknowledgement data message in detailed format.
Data Type: LongDAAmsg
Privilege: Private
Default Value:

Operations:

FillDAA - This function will package a long DAA message for the DAN acknowledgement.
Arguments: int Status[],char *DataType[],char *Descriptor, int FileGroupCount, int DANSeqNo
Return Type: Void
Privilege: Public

Associations:

The InLongDAA class has associations with the following classes:
None

4.3.23 InLongDDN Class

Parent Class: InMessage
Public: No
Distributed Object: No
Purpose and Description:
This object class populates the long DDN (Data Delivery Notice) data message to be sent to the external Client after the data is archived.

Attributes:

myLongDDN - This is the long DDN (Data Delivery Notice) data message.
Data Type: LongDDNmsg
Privilege: Private
Default Value:

Operations:

FillDDN - Packages the DDN data message with given inputs.
Arguments: int Status[], char *Direcotry, char *FileId, int FileCount, int DANSeqNo
Return Type: Void
Privilege: Public

Associations:

The InLongDDN class has associations with the following classes:
None

4.3.24 InMediaIngest Class

Parent Class: InGUISession
Public: No
Distributed Object: No
Persistent Class:
Purpose and Description:
Provides operations personnel the capability to perform physical media ingest via the GUI interface. The is a derived object class from the InGUISession object class. It inherits all data and service members provided by the InGUISession.

Attributes:

All Attributes inherited from parent class

Operations:

CheckPrivilege - Verifies that the operator and the media provider has ingest privilege.
Arguments: char *UserName
Return Type: int
Privilege: Public

Associations:

The InMediaIngest class has associations with the following classes:
None

4.3.25 InMessage Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Purpose and Description:
Contains data messages that interchanges between the external Client and ECS/Ingest.

Attributes:

None

Operations:

GetMsgLength - Extracts and returns the message type from the data message.
Arguments: char *MsgPtr
Return Type: Void
Privilege: Public

InMessage - This is the constructor for the InMessage object class.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The InMessage class has associations with the following classes:
Class: InSession Receives/Sends - The InSession object interfaces with the InMessage object to access data messages that are interchanged between Ingest and the external Client.

4.3.26 InMetadata Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Purpose and Description:
This is an abstract class.

Attributes:

myInFile - This attribute defines the associated input files.
Data Type: char*
Privilege: Private
Default Value:

myInSourceMCF - This attribute specifies the associated source metadata configuration file.
Data Type: char*
Privilege: Private
Default Value:

Operations:

Preprocess - This is an abstract operation.
Arguments:

Associations:

The InMetadata class has associations with the following classes:
Class: InSourceMCF defines - The InSourceMCF class defines the InMetadata class by providing format informatin on input metadata files.
Class: DsCIDescriptor guides - The DsCIDescriptor class guides the InMetadata class by providing services to access target MCFs and validate metadata files.
Class: InMetadataTool utilizes - The InMetadata class utilizes the InMetadataTool to read targetMCFsand write PVL metadata.
InDataType (Aggregation)

4.3.27 InMetadataTool Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Purpose and Description:
This class provides services to read the parameters in a target metadata configuration file, set the values of these parameters, and the write the modified target MCF to a final PVL file.

Attributes:

mypthread - This attribute defines the thread the metadata tool object is associated with.
Data Type: int
Privilege: Private
Default Value:

Operations:

PGS_MET_GetNext - Provides the next parameter name within the targetMF.
Arguments:
Return Type: char* parameter
Privilege: Public

PGS_MET_INIT - Initializes a metadata configuration file. The contents of the metadata configuration file are read into memory to provide a basis for setting and checking metadata parameter values.
Arguments:

PGS_MET_Set - The value of any of the parameters from the target MCF can be set using this operation.
Arguments:

PGS_MET_WriteFile - Once the parameter values have been set in the Target MCF, this operation writes the contents of the MCF memory section out to a physical file.
Arguments:

Associations:

The InMetadataTool class has associations with the following classes:
Class: InFile manipulates - The InMetadata class manipulates the InFile class by reading/writing files.
Class: InMetadata utilizes - The InMetadata class utilizes the InMetadataTool to read targetMCFsand write PVL metadata.

4.3.28 InNetworkIngest Class

Parent Class: InGUISession
Public: No
Distributed Object: No
Persistent Class:
Purpose and Description:
Provides authorized science users the capability to ingest data electronically via the GUI interface. The object class is derived from the InGUISession object class. It inherits all the data and service members provided by the InGUISession object class.

Attributes:

All Attributes inherited from parent class

Operations:

SaveRequestToFile - Saves the network ingest request contents on the screen onto a user specified file name.
Arguments: char *FileName
Return Type: Void
Privilege: Public

Associations:

The InNetworkIngest class has associations with the following classes:
None

4.3.29 InPVMetadata Class

Parent Class: InMetadata
Public: No
Distributed Object: No
Purpose and Description:
This class will provides services to preprocess Parameter-Value metadata.

Attributes:

myLineDelimiter - This attribute will define the symbol used to indicate the end of a parameter-value metadata statement.
Data Type: char*
Privilege: Private
Default Value:

myParameterDelimiter - This attribute will define the symbol used to delimit the parameter portion of a parameter-value metadata statement.
Data Type: char*
Privilege: Private
Default Value:

mySeparator - This attribute will define the symbol used to separate the parameter from the value in the parameter-value metadata statement
Data Type: char*
Privilege: Private
Default Value:

myValueDelimiter - This attribute will define the symbol used to delimit the value part of a parameter-value metadata statement.
Data Type: char*
Privilege: Private
Default Value:

Operations:

InPVMetadata - This is the constructor service.
Arguments:

Preprocess - This operation will extract values from input parameter-value files based on delimiters and parameter names. It will access the correct MCFs and interact with the InMetadata_Tool Class to produce a metadata file which is acceptable to the data server subsystem.
Arguments:
Return Type: char* status
Privilege: Public

Associations:

The InPVMetadata class has associations with the following classes:
None

4.3.30 InPollingIngestSession Class

Parent Class: InSession
Public: No
Distributed Object: No
Persistent Class:
Purpose and Description:
This object class does not have any control link with the external interface (i.e, no physical stimulus provided from external source). It is a persistent object class which is configured to wake up at a tunable period of time to detect existence of ingest files at a designated location; the location could either be external or local in ECS. If files are detected, the object class will instantiate the InPollingIngestRequest object class and add it to the InRequestList to be processed. The InPollingIngestSession object class is derived from the InSession. It inherits all the data and service members provided by the InSession object class.

Attributes:

myPollingTimer - The time period which indicates how often the Polling Ingest Session should check for the existence of ingest files for ingest processing.
Data Type: int
Privilege: Private
Default Value:

Operations:

CleanupDirectory - Peforms directory cleanup after files are ingested by means of moving the complete files to another directory so that these files will not be picked up again for the next ingesting.
Arguments: void
Return Type: int
Privilege: Public

DeliverResponse - Generates the ingest response pertaining to the ingest polling process in a file.
Arguments: char *ResponseFile
Return Type: Void

Privilege: Public

ProcessRequest
Arguments: char * FileInfo
Return Type: int
Privilege: Public

Associations:

The InPollingIngestSession class has associations with the following classes:
None

4.3.31 InPollingThreshold Class

Parent Class: InExternalDataProviderThreshold
Public: No
Distributed Object: No
Persistent Class: True
Purpose and Description:
This is a persistant object class that defines thresholds for the Ingest Polling Interface.

Attributes:

myPollingTimer - Indicates the time period to wait before starting the Ingest Polling.
Data Type: int
Privilege: Private
Default Value:

Operations:

GetTimer - Get the polling timer for the associated external client.
Arguments: void
Return Type: int
Privilege: Public

SetTimer - Sets the polling timer for the associated external client.
Arguments: int
Return Type: int

Privilege: Public

Associations:

The InPollingThreshold class has associations with the following classes:
None

4.3.32 InReformatData Class

Parent Class: InScienceData
Public: No
Distributed Object: No
Purpose and Description:
This class provides services to preprocess data which is not in an ECS compatible format.

Attributes:

All Attributes inherited from parent class

Operations:

InReformatData - This is the constructor service.
Arguments:

Preprocess - This operation will reformat input science data files which are not in an acceptable ECS data format. The reformatting includes byte swapping and other functions to resolve platform incompatibilities.
Arguments:
Return Type: char* Status
Privilege: Public

Associations:

The InReformatData class has associations with the following classes:
None

4.3.33 InRequest Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class:
Purpose and Description:
Contains information provided by a requestor/external interface requesting ingest of data. The object class has the responsibility to perform basic request component checking and to assign a unique identifier for the new ingest request.

Attributes:

myAggregateLength - Total volume of data (in bytes) to be ingested based on the given request.
Data Type: int
Privilege: Private
Default Value:

myDataTypeIdList - The set of data types associate with the ingest files.
Data Type: struct **
Privilege: Private
Default Value:

myExpirationDateTime - Date/time by which the corresponding ingest request must be completed (i.e., archive insertion complete and response returned to the external data provider).
Data Type: char*
Privilege: Private
Default Value:

myExternalDataProvider - Identifier of the external data source providing data to be ingested into ECS.
Data Type: char*
Privilege: Private
Default Value:

myIngestType - The type of data ingest to be performed (e.g., media ingest, network ingest).
Data Type: char*
Privilege: Private
Default Value:

myProcessingEndDateTime - Ending date/time (in standard ECS date/time format) at

which the ingest request processing completed (the time immediately prior to deleting the object in the destructor service).
Data Type: DateTime
Privilege: Private
Default Value:

myProcessingStartDateTime - Starting date/time (in standard ECS time format) at which ingest processing began (time of creation of the InRequest object).
Data Type: DateTime
Privilege: Private
Default Value:

myRequestId - The information that uniquely identifies an Ingest Request. Request Identifiers are referenced by Status Requests and other Service Requests that are used to monitor or control the execution of Ingest Requests.
Data Type: int
Privilege: Private
Default Value:

myRequestPriority - The information that determines the order in which an ingest request will be processed relative to other ingest requests waiting to be processed. The priority is provided by the InExternalDataProvider object class for each external data provider.
Data Type: int
Privilege: Private
Default Value:

myRequestState - State of the corresponding ingest request. Values are "Active" and "Complete".
Data Type: char*
Privilege: Private
Default Value:

mySequenceId - The SequenceId identifies each of the control messages for a given request. The sequence number is first extracted from the DAN, then all the control messages (e.g, DAA,DRA,DRR...) need to contain the same sequence number for a given request.
Data Type: int
Privilege: Private
Default Value:

mySessionId - The identifier of the session associated with the ingest request.
Data Type: int
Privilege: Private
Default Value:

myTotalFileCount - Total number of files identified for ingest in the request.
Data Type: int
Privilege: Private
Default Value:

Operations:

Cancel - Cancels an ingest request.
Arguments: void
Return Type: int
Privilege: Public

ChangeState - Updates the state of an ingest request.
Arguments: String *NewState
Return Type: int
Privilege: Private

Check - Verifies the components in the Request.
Arguments: void
Return Type: int
Privilege: Public

GetRequestId - Assigns an unique identifier for the ingest request.
Arguments: void
Return Type: int
Privilege: Private

GetSessionId - Returns the identifier of the Session that the ingest request is running.
Arguments: void
Return Type: int
Privilege: Public

GetState - Returns the State that the Session is on.
Arguments: void
Return Type: char*
Privilege: Private

InRequest - Constructor, when a DAN file is supplied.
Arguments: char* DANfile
Return Type: int
Privilege: Public

InRequest - Constructor, when a DAN message is supplied.

Arguments: DANmsg *DANmsgPtr
Return Type: int
Privilege: Public

ProcessRequest - Service to invoke ingest processing (data transfer, data preprocessing, data insertion) for a given ingest request.
Arguments: void
Return Type: int
Privilege: Public

Associations:

The InRequest class has associations with the following classes:

Class: InSession Creates - An instance of InSession object could create one or more instance of InRequest objects.
Class: InDataTransferTask InitiatesDataTransferVia - The InRequest object interfaces with the InDataTransferTask object to perform data transfer.
Class: InRequestManager IsManagedBy - InRequestManager accesses InRequestList.
Class: InDataPreprocessTask IsPreprocessedBy - An instance of the InRequest object interfaces with one or more instances of InDataPreprocessTask object for data preprocessing.
Class: InDAN IsStoredIn - The InDAN object information is stored in InRequest.
Class: InDataServerInsertionTask SendsInsertRequestVia - The InRequest class SendsInsertRequestVia the InDataServerInsertionTask class to initiate the process necessary to insert data into the Data Server Subsystem.
Class: InRequestProcessData IsStoredIn - InRequestProcessData object checkpoints InRequest.
Class: InRequestSummaryData IsStoredIn - InRequestSummaryData object checkpoints InRequest.
Class: InRequestSummaryHeader IsStoredIn - InRequestSummaryHeader object checkpoints InRequest.
Class: InRequestProcessHeader Stores - InRequestProcessHeader object checkpoints InRequest.

4.3.34 InRequestController Class

Parent Class: InGUISession
Public: No
Distributed Object: No
Persistent Class:
Purpose and Description:

Provides authorized operations personnel the capability to udpate an ongoing ingest request via the GUI interface. The operations personnel could 1) cancel an ingest request, 2) suspend an ingest request, 3) resume an ingest request, or 4) change priority of an ingest request. This is a derived object class from the InGUISession object class. It inherits all the data and services provided by the InGUISession object class.

Attributes:

myRequestCriteria - The criteria spcified by the Operator for rsearching the appropriate requests which are to be updated.

myRequestId - The identifier of the ingest request that is to be updated (e.g., cancel, suspend, resume, change priority).
Data Type: int
Privilege: Private
Default Value:

myRequestIdList - List of identifier of requests to be updated.

myUpdateType - Identifies the type of update to be performed on an ingest request. The types of ingest request updates consist of: cancel, suspend, resume, and change priority.
Data Type: int
Privilege: Private
Default Value:

Operations:

ProcessRequest - Invokes appropriate services to perform the update service on the specified ingest request. This service overloads the ProcessRequest() service defined the InGUISession object class.
Arguments: int RequestId, int UpdateType
Return Type: Void
Privilege: Public

Associations:

The InRequestController class has associations with the following classes:
Class: InRequestList UpdatesRequestFrom - The InRequestController interfaces with the InRequestList object to locate the InRequest object on which the update is to be performed

(e.g., change priority, cancel, suspend, resume).

4.3.35 InRequestFileInfo Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class:
Purpose and Description:
Provides checkpoint storage of file information associated with a given data granule in a given ingest request. In the event of request completion, this item is deleted.

Attributes:

myBeginningDateTime - The starting date/time of the file ingest.
Data Type: DateTime
Privilege: Private
Default Value:

myDirectoryID - The directory containing the Request files.
Data Type: char*
Privilege: Private
Default Value:

myFileID - The identifier of the file.
Data Type: char*
Privilege: Private
Default Value:

myFileSize - The size of ingest file.
Data Type: int
Privilege: Private
Default Value:

myRecordSize - Indicates the file record size.
Data Type: int
Privilege: Private
Default Value:

myRequestID - Identifier of the InRequest_S object to which this entry corresponds. This is a primary key.
Data Type: char*

Privilege: Private
Default Value:

Operations:

InRequestFileInfo - This is the object construct. It populates the object and the associated data base table entry.
Arguments: void
Return Type: int
Privilege: Public

SearchTable - Locates a stored entry based on Request ID.
Arguments: char* myRequestID
Return Type: int
Privilege: Public

~InRequestFileInfo - This is the object destructor. It deletes the object and the associated data base table entry.
Arguments: char* myRequestID
Return Type: int
Privilege: Public

Associations:

The InRequestFileInfo class has associations with the following classes:
InRequestProcessData (Aggregation)

4.3.36 InRequestInfo Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class: True
Purpose and Description:
Keeps track of all the requests running per session. Each request information is inserted upon receipt of DAN, and deleted from the list upon completion of request (receipt of DAA).

Attributes:

DANSeqNum - DAN Sequence number associated with the request.
Data Type: int
Privilege: Private
Default Value:

RequestId - The identifer of the request.
Data Type: int
Privilege: Private
Default Value:

Operations:

AddRequest - Inserts a new request information into the Request List of the InSession.
Arguments: int RequestId, int DANseqNum
Return Type: Void
Privilege: Public

DeleteRequest - Deletes request information from the Request List of the InSession.
Arguments: int DANSeqNum
Return Type: Void
Privilege: Public

GetRequestCount - Get the total number of requests in the Request List of the InSession.
Arguments: void
Return Type: Void
Privilege: Public

InRequestInfo - This is the constructor of the InRequestInfo object class.
Arguments: int SeqNum, int ReqId
Return Type: Void
Privilege: Public

ListRequests - Lists all the request in the Request List of the Session.
Arguments: void
Return Type: Void
Privilege: Public

SearchRequest - Searches for a request in the list with the DAN sequence number.
Arguments: int DANSeqNum
Return Type: Void

Privilege: Public

Associations:

The InRequestInfo class has associations with the following classes:
Class: InSession Manages - The InSession object class manages the InRequestInfo object class.

4.3.37 InRequestList Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class: True
Purpose and Description:
Contains a list of all the ingest requests that are currently ongoing or waiting to be processed in the order based on the priority. The object class has the responsibility 1) to add ingest request to the list based on the priority, 2) to delete an ingest request, 3) to search for an ingest request, and 4) to list the ingest requests.

Attributes:

myCurrentPointer - The current position in the Ingest Request List.
Data Type: int *CurPointer
Privilege: Private
Default Value:

myListCounter - The number of requests in the Ingest Request List.
Data Type: int Counter
Privilege: Private
Default Value:

myListHead - The first request in the Ingest Request List.
Data Type: int *StartPointer
Privilege: Private
Default Value:

myListTail - The last request in the Ingest Request List.
Data Type: int EndPointer
Privilege: Private

Default Value:

Operations:

AddRequest - Adds a request entry to the list.
Arguments: struc*Request
Return Type: int
Privilege: Public

DeleteRequest - Deletes a request entry from the list.
Arguments: int RequestId
Return Type: int
Privilege: Public

GetNext - Get the next request entry from the list.
Arguments: int RequestId
Return Type: int
Privilege: Public

ListAll - Lists all request entries in the list.
Arguments: viod
Return Type: int
Privilege: Public

SearchRequest - Searches for a request entry on the list using Request ID.
Arguments: int RequestId
Return Type: int
Privilege: Public

Associations:

The InRequestList class has associations with the following classes:
Class: InStatusMonitor QueriesRequest(s)From - The InStatusMontiro object interfaces with InRequestList to locate InReqeust(s) for status monitoring.
Class: InRequestController UpdatesRequestFrom - The InRequestController interfaces with the InRequestList object to locate the InRequest object on which the update is to be performed (e.g., change priority, cancel, suspend, resume).

4.3.38 InRequestManager Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class:
Purpose and Description:
This is a focal object class of the Ingest CI. It coordinates the ingest processing which includes the initiating of the data transfer and the sending of data insertion request to the appropriate Data Server. The object class also tracks and allows updates the ingest thresholds.

Attributes:

myCurrentDataVolumeKeep - Current running total of data volume in active ingest request. Incremented as new InRequest objects are created; decremented when InRequest objects are deleted.
Data Type: int
Privilege: Private
Default Value:

myCurrentRequests - Keeps a running total of the number of requests currently in the system. Incremented as new InRequest objects are created; decremented when InRequest objects are deleted.
Data Type: int
Privilege: Private
Default Value:

Operations:

CancelRequest - Requests cancellation of an existing InRequest object. The InRequest CancelRequest service is invoked. The success of the cancellation request depends on the state of the ongoing request.
Arguments: DCEObjRefT* ObjReference
Return Type: int
Privilege: Public

CreateRequest - Creates a new InRequest object in a pthead when a DAN file pointer is provided.
Arguments: char* DANfile
Return Type: DCEObjRefT*
Privilege: Public

CreateRequest - Creates a new InRequest object in a pthread when a DAN message pointer is provided.
Arguments: DANmsg* DANmsgPtr
Return Type: DCEObjRefT*
Privilege: Public

DeleteRequest - Deletes a currently instantiated InRequest object.
Arguments: DCEObjRefT* ObjReference
Return Type: int
Privilege: Public

GetCurrentDataVolume - Gets the value of the current data volume.
Arguments: void
Return Type: int
Privilege: Public

GetCurrentRequests - Gets the value of the current total number of ingest requests in the system.
Arguments: void
Return Type: int
Privilege: Public

RestoreRequestList - Recovers the InRequestList object after a process or system failure.
Arguments: void
Return Type: int
Privilege: Private

UpdateCurrentDataVolume - Updates the value of the current running total of data volume requested to be ingested.
Arguments: int UpdateValue
Return Type: int
Privilege: Private

UpdateCurrentRequests - Updates the current running total for number of ingest requests in the system.
Arguments: int UpdateValue
Return Type: int
Privilege: Private

Associations:

The InRequestManager class has associations with the following classes:
Class: InRequestList IsAccessedBy - InRequestManager accesses InRequestList.
Class: InRequest IsManagedBy - InRequestManager accesses InRequestList.
Class: InThreshold ProvidesThresholdsfor - InRequestManager retries system and external data provider based thresholds for ingest processing.

4.3.39 InRequestManager_C Class

Parent Class: Not Applicable
Public: No
Distributed Object: Yes
Purpose and Description:
The client implementation of the distributed InRequestManager object. This client acts as the intermediary to the object factory (InRequestManager_S).

Attributes:

None

Operations:

CreateRequest - Service to create an InRequest object via a distributed object factory (InRequestManager_S) when a DAN message is supplied.
Arguments: DANmsg* DANmsgPtr
Return Type: int
Privilege: Public

CreateRequest - Service to create an InRequest object via a distributed object factory (InRequestManager_S) when a DAN file is supplied.
Arguments: char* DANfile
Return Type: int
Privilege: Public

Associations:

The InRequestManager_C class has associations with the following classes:
InRequestManager (Aggregation)

4.3.40 InRequestManager_S Class

Parent Class: Not Applicable
Public: No
Distributed Object: Yes
Purpose and Description:
Implementation of the server (object factory) for InRequestManager.

Attributes:

None

Operations:

CreateRequest - Service to create an InRequest_S object when a DAN message is supplied.
Arguments: DANmsg* DANmsgPtr
Return Type: int
Privilege: Public

CreateRequest - Service to create an InRequest_S object when a DAN file is supplied.
Arguments: char* DANfile
Return Type: int
Privilege: Public

Associations:

The InRequestManager_S class has associations with the following classes:
InRequestManager (Aggregation)

4.3.41 InRequestProcessData Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class: True
Purpose and Description:
Provides checkpoint storage of data granule processing information associated with a given ingest request. In the event of request completion, this item is deleted.

Attributes:

myDataDescriptor - ASCII description of the data type.
Data Type: char*
Privilege: Private
Default Value:

myDataGranuleID - Numeric (ASCII) identifier of a data granule within an ingest request. Determined incrementally for each data granule in an ingest request.
Data Type: char*
Privilege: Private
Default Value:

myDataGranuleState - State of processing ("Not transferred", "Transferred", "Submitted", and "Inserted") of a data granule for a given ingest request.
Data Type: char*
Privilege: Private
Default Value:

myDataVersion - Version number of the data granule to be ingested (> 1 if a reprocessed granule).
Data Type: char*
Privilege: Private
Default Value:

myRequestId - Identifier of the InRequest_S object to which this entry corresponds. This is a primary key.
Data Type: char*
Privilege: Private
Default Value:

Operations:

InRequestProcessData - Constructor; populates the object and the associated data base table entry.
Arguments: void
Return Type: int
Privilege: Public

SearchTable - Locates a stored entry based on Request ID.
Arguments: char* myRequestID
Return Type: int
Privilege: Public

~InRequestProcessData - Destructor; deltes the object and the associated data base table entry.
Arguments: char* myRequestID
Return Type: int
Privilege: Public

Associations:

The InRequestProcessData class has associations with the following classes:
Class: InRequest IsStoredIn - InRequestProcessData object checkpoints InRequest.

4.3.42 InRequestProcessHeader Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class: True
Purpose and Description:
Provides checkpoint storage of ingest request processing information associated with a given ingest request. In the event of request completion, this item is deleted.

Attributes:

myExpirationDateTime - Date/time by which the corresponding ingest request must be completed (i.e., archive insertion complete and response returned to the external data provider).
Data Type: DateTime
Privilege: Private
Default Value:

myRequestID - Identifier of the InRequest_S object to which this entry corresponds. This is a primary key.
Data Type: char*
Privilege: Private
Default Value:

myRequestPriority - The information that determines the order in which an ingest request will be processed relative to other ingest requests waiting to be processed. The priority is provided by the InExternalDataProvider object class for each external data provider.

Data Type: int
Privilege: Private
Default Value:

myRequestState - State of the corresponding ingest request. Values are "Active" and "Complete".
Data Type: char*
Privilege: Private
Default Value:

mySequenceID - The SequenceId identifies each of the control messages for a given request. The sequence number is first extracted from the DAN, then all the control messages (e.g, DAA,DRA,DRR...) need to contain the same sequence number for a given request.
Data Type: int
Privilege: Private
Default Value:

mySessionID - Identifier of the InSession object to which the InRequest object corresponding to this entry is associated.
Data Type: char*
Privilege: Private
Default Value:

Operations:

InRequestProcessHeader - Constructor; populates the object and the associated data base table entry.
Arguments: void
Return Type: int
Privilege: Public

SearchTable - Locates a stored entry based on Request ID.
Arguments: char* myRequestID
Return Type: int
Privilege: Public

~InRequestProcessHeader - Destructor; deletes the object and the associated data base table entry.
Arguments: char* myRequestID
Return Type: int

Privilege: Public

Associations:

The InRequestProcessHeader class has associations with the following classes:
Class: InRequest Stores - InRequestProcessHeader object checkpoints InRequest.

4.3.43 InRequestSummaryData Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class: True
Purpose and Description:
Provides long-term storage of summary data type statistics associated with a given data granule in a given ingest request.

Attributes:

myDataGranuleID - Numeric (ASCII) identifier of a data granule within an ingest request. Determined incrementally for each data granule in an ingest request.
Data Type: char*
Privilege: Private
Default Value:

myDataGranuleVolume - Total data volume to be ingested for a data granule in an ingest request. The total data volume for the data granule is determined by summing the data volumes for the files comprising the data granule.
Data Type: int
Privilege: Private
Default Value:

myDataType - Data type identifier for the data granule. Selected from a list of valid data type identifiers maintained by the Data Server.
Data Type: char*
Privilege: Private
Default Value:

myFinalStatus - Final error status for the ingest processing of a data granule.
Data Type: int

Privilege: Private
Default Value:

myRequestID - Identifier of the InRequest_S object to which this entry corresponds. This is a primary key.
Data Type: char*
Privilege: Private
Default Value:

Operations:

InRequestSummaryData - Constructor; populates the object and the associated data base entry.
Arguments: void
Return Type: int
Privilege: Public

SearchTable - Locates a stored entry based on Request ID.
Arguments: char *myRequestID
Return Type: int
Privilege: Public

~InRequestSummaryData - Destructor; deletes the object and the associated data base table entry.
Arguments: char* myRequestID
Return Type: int
Privilege: Public

Associations:

The InRequestSummaryData class has associations with the following classes:
Class: InRequest IsStoredIn - InRequestSummaryData object checkpoints InRequest.

4.3.44 InRequestSummaryHeader Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class: True

Purpose and Description:
Provides long-term storage of summary request-level statistics associated with a given ingest request.

Attributes:

myAggregateLength - Total volume of data (in bytes) to be ingested based on the given request.
Data Type: int
Privilege: Private
Default Value:

myExternalDataProvider - Identifier of the external data provider (e.g., TSDIS) associated with an ingest request.
Data Type: char*
Privilege: Private
Default Value:

myMission - Name of the science mission (e.g., Landsat 7) for which the data described in the request was generated.
Data Type: char*
Privilege: Private
Default Value:

myProcessingEndDateTime - Ending date/time (in standard ECS date/time format) at which the ingest request processing completed (the time immediately prior to deleting the object in the destructor service).
Data Type: DateTime
Privilege: Private
Default Value:

myProcessingStartDateTime - Starting date/time (in standard ECS time format) at which ingest processing began (time of creation of the InRequest object).
Data Type: DateTime
Privilege: Private
Default Value:

myRequestID - Identifier of the InRequest_S object to which this entry corresponds. This is a primary key.
Data Type: char*
Privilege: Private
Default Value:

myTotalDataGranules - Total number of granules associated with an ingest request. This

value is determined by counting the number of data granule entries in the ingest request.
Data Type: int
Privilege: Private
Default Value:

myTotalFileCount - Total number of files identified for ingest in the request.
Data Type: int
Privilege: Private
Default Value:

Operations:

InRequestSummaryHeader - Constructor; populates the object and the associated data base table entry.
Arguments: void
Return Type: int
Privilege: Public

SearchTable - Locates a stored entry based on Request ID.
Arguments: char *myRequestID
Return Type: int
Privilege: Public

~InRequestSummaryHeader - Destructor; deletes the object and the associated data base table entry.
Arguments: char* myRequestID
Return Type: int
Privilege: Public

Associations:

The InRequestSummaryHeader class has associations with the following classes:
Class: InRequest IsStoredIn - InRequestSummaryHeader object checkpoints InRequest.

4.3.45 InRequest_C Class

Parent Class: Not Applicable
Public: No
Distributed Object: Yes

Purpose and Description:
This is the client implementation of the InRequest distributed object. Generated by IDL.

Attributes:

myRequestID - Identifier of the InRequest object to which this client object communicates.
Data Type: char *
Privilege: Private
Default Value:

Operations:

InRequest_C - The constructor, where a DAN message structure is entered.
Arguments: DANmsg* DANmsgPtr
Return Type: int
Privilege: Public

InRequest_C - The constructor, where a DAN file is specified.
Arguments: char* DANfile
Return Type: int
Privilege: Public

Associations:

The InRequest_C class has associations with the following classes:
InRequest (Aggregation)

4.3.46 InRequest_S Class

Parent Class: Not Applicable
Public: No
Distributed Object: Yes
Purpose and Description:
This is the server implementation of the InRequest distributed object. Generated from IDL.

Attributes:

myRequestID - Identifier of the InRequest object to which this server object communicates.
Data Type: char *
Privilege: Private
Default Value:

Operations:

InRequest_S - Constructor, where a DAN message is supplied.
Arguments: DANmsg* DANmsgPtr
Return Type: int
Privilege: Public

InRequest_S - Constructor, where a DAN file is supplied.
Arguments: char* DANfile
Return Type: int
Privilege: Public

Associations:

The InRequest_S class has associations with the following classes:
InRequest (Aggregation)

4.3.47 InResourceIF Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class:
Purpose and Description:
Serves as an interface to resource/device services (e.g., device allocation, data transferring, and file information) by which the Data Server Subsystem provides. The object class interfaces with the Storage Resource Management and Data Distribution CSCIs of the Data Server Subsystem. Refer to the Data Server Subsystem section of this document for details.

Attributes:

myResourceId - The identifier of the allocated resource.
Data Type: int
Privilege: Private

Default Value:

myResourceType - The type of resource (e.g., working storage, physical media) to be used for the Ingest processing.
Data Type: int
Privilege: Private
Default Value:

Operations:

AllocateResource - Allocates for an available Resource device.
Arguments: int ResourceId
Return Type: int
Privilege: Public

CopyFile - Copies a file from the specified resource to the specified directory.
Arguments: void
Return Type: int
Privilege: Public

DeallocateResource - Deallocates the resource device.
Arguments: int ResourceId
Return Type: int
Privilege: Public

GetFileInfo - Gets the file information on the resource.
Arguments: Char *FileId, char *FileLocation, int FileSize
Return Type: int
Privilege: Public

ListFiles - Lists all the files in the resource.
Arguments: void
Return Type: int
Privilege: Public

Associations:

The InResourceIF class has associations with the following classes:
Class: InDataTransferTask AllocatesResourceFrom,TransfersFilesFrom - The InDataTransferTask object performs device allocation and bulk data transfer via the InResourceIF object.
Class: InFile ResidesOn - The InFile object performs the file transfer and the file

transmission check via the InResourceIF object.

4.3.48 InSDMetadata Class

Parent Class: InMetadata

Public: No

Distributed Object: No

Purpose and Description:

This class provides services to preprocess data which is in self-descriptive format other than HDF.

Attributes:

All Attributes inherited from parent class

Operations:

InSDMetadata - This is the constructor service.

Arguments:

Preprocess - This operation will extract values from input files with a self-descriptive format (other than HDF). It will access the correct MCFs and interact with InMetadataTool Class to produce a metadata file which is acceptable to the Data Server Subsytem.

Arguments:

Return Type: char* status

Privilege: Public

Associations:

The InSDMetadata class has associations with the following classes:

None

4.3.49 InScienceData Class

Parent Class: Not Applicable

Public: No

Distributed Object: No

Purpose and Description:
This is an abstact class

Attributes:

myInFile - This attribute defines the associated input file.
Data Type: char*
Privilege: Private
Default Value:

Operations:

Preprocess - This is an abstact operation.
Arguments:

Associations:

The InScienceData class has associations with the following classes:
InDataType (Aggregation)

4.3.50 InServer Class

Parent Class: Not Applicable
Public: No
Distributed Object: Yes
Persistent Class:
Purpose and Description:
Provides a single point of entry to the Ingest system for all ingest interfaces. The object class manages ingest sessions.

Attributes:

mySessionCount - The total number of sessions running under the Ingest Server.
Data Type: int
Privilege: Private
Default Value:

Operations:

StartServer - Starts up the Ingest Server.
Arguments: void
Return Type: int
Privilege: Public

Associations:

The InServer class has associations with the following classes:
Class: InSessionInfo IsManagedBy
Class: InSession Manages - All instance of InSession object is managed by one instance of InServer object.

4.3.51 InServerExtRPC_C Class

Parent Class: Not Applicable
Public: Yes
Distributed Object: Yes
Purpose and Description:
This is the client/proxy implementation that defines the RPC for initiating an Ingest Session.

Attributes:

None

Operations:

CreateSession - This is a RPC that initiates a new Ingest Session.
Arguments: handle_t InServerBH, char *GatewayStringBH, error_status_t *CreateSessStatus
Return Type: int
Privilege: Public

Associations:

The InServerExtRPC_C class has associations with the following classes:

Class: CsGateWay IsInvokedBy - The Gateway object interfaces with the InServerExtRPC to initiate a new Ingest Session through the Ingest Server.

InServer (Aggregation)

4.3.52 InServerExtRPC_S Class

Parent Class: Not Applicable

Public: No

Distributed Object: Yes

Purpose and Description:

This is the server implementation that defines the services for creating a new session.

Attributes:

None

Operations:

CreateSession - Creates a new session for a given client upon receipt of an Authentication Request.

Arguments: handle_t InServerBH, char *GatewayStringBH, error_status_t *CreateSessStatus

Return Type: Void

Privilege: Public

Associations:

The InServerExtRPC_S class has associations with the following classes:

InServer (Aggregation)

4.3.53 InServerIntRPC_C Class

Parent Class: Not Applicable

Public: Yes

Distributed Object: Yes

Purpose and Description:

This is the client/proxy implmentation for the InServer object class. The provided services are to be used by the InSession object class.

Attributes:

None

Operations:

DeleteSession - Deletes the specified session from the InServer's Session List.
Arguments: handle_t InServerBH, int SessionId, error_status_t *DelSessStatus
Return Type: Void
Privilege: Public

Associations:

The InServerIntRPC_C class has associations with the following classes:
Class: InSession IsInvokedBy - InSession intefaces with InServerIntRPC_C to delete itself from the InServer's session list.
InServer (Aggregation)

4.3.54 InServerIntRPC_S Class

Parent Class: Not Applicable
Public: No
Distributed Object: Yes
Purpose and Description:
This is the server implementation (factory) for the InServer object class. The provides services are to be used by the InSession object class.

Attributes:

None

Operations:

DeleteSession - Deletes the specified session from the InServer's Session List.
Arguments: handle_t InServerBH, int SessionId, error_status_t *DelSessStatus
Return Type: Void

Privilege: Public

Associations:

The InServerIntRPC_S class has associations with the following classes:
InServer (Aggregation)

4.3.55 InSession Class

Parent Class: Not Applicable
Public: No
Distributed Object: Yes
Persistent Class:
Purpose and Description:
This is the super object class for specialization object classes that handle specific external interfaces. In general, the object class manages the hand-shaking protocal with the ingest service requestor. It verifies that the requestor has privilege to perform the data ingest service. The InSession instantiates the InRequest and adds to the InRequestList to be processed. In addition, the InSession allows cancellation, suspension, and resumption of the Ingest Request processing running under the session. Suspension and resumption are post Release A functions.

Attributes:

myClientId - Session's client identifier.
Data Type: char *
Privilege: Private
Default Value:

mySessionGWBH - Session's binding handle with the Gateway.
Data Type: char *
Privilege: Private
Default Value:

mySessionId - The information that uniquely identifies the session.
Data Type: int
Privilege: Private
Default Value:

Operations:

InitSessServer - Starts up the session. The operation is invoked by the Ingest Server upon receipt of an Create Session request from Gateway.
Arguments: char *GatewayBH
Return Type: int
Privilege: Public

ProcessRequest - Once DAN is received from the Client, this operation instantiates a new Request an adds the request to the Ingest Request List, and sends DAA (DAN Acknowledgement) to the Client.
Arguments: void
Return Type: int
Privilege: Public

ResumeSession - Resumes the session. All ingest processing running under the session will be resumed. This is a post Relase A service.
Arguments: void
Return Type: int
Privilege: Public

SuspendSession - Suspends the session. All ingest processing running under the session will be suspended. This is a post Release A service.
Arguments: void
Return Type: int
Privilege: Public

TerminateSession - Terminates the session. All ingest processing running under the session will be terminated.
Arguments: void
Return Type: int
Privilege: Public

Associations:

The InSession class has associations with the following classes:
Class: InRequest Creates - An instance of InSession object could create one or more instance of InRequest objects.
Class: InServerIntRPC_C IsInvokedBy - InSession intefaces with InServerIntRPC_C to delete itself from the InServer's session list.
Class: InSessionEcsRPC_C IsInvokedBy
Class: InRequestInfo Manages - The InSession object class manages the InRequestInfo object class.

Class: InServer Manages - All instance of InSession object is managed by one instance of InServer object.
Class: InMessage Receives/Sends - The InSession object interfaces with the InMessage object to access data messages that are interchanged between Ingest and the external Client.

4.3.56 InSessionEcsRPC_C Class

Parent Class: Not Applicable
Public: Yes
Distributed Object: Yes
Purpose and Description:
This is the client/proxy implementation that defines services for sending outgoing data messages from the ECS Ingest.

Attributes:

None

Operations:

ecsDDN - The RPC is invoked by ECS Ingest to send the Data Delivery Notice (DDN) data message to the external client.
Arguments: handle_t GatewayBH, char *DDN, error_status_t ecsDDNstatus
Return Type: Void
Privilege: Public

Associations:

The InSessionEcsRPC_C class has associations with the following classes:
Class: InSession IsInvokedBy
CsGateWay (Aggregation)

4.3.57 InSessionEcsRPC_S Class

Parent Class: Not Applicable
Public: No
Distributed Object: Yes
Purpose and Description:

This is the server implementation that defines services for sending outgoing data messages from ECS Ingest to external client.

Attributes:

None

Operations:

ecsDDN - The RPC is invoked by ECS Ingest to send DDN (Data Delivery Notice) data message to the external client.
Arguments: handle_t GatewayBH, char *DDNmsg, error_status_t ecsDDNstatus
Return Type: Void
Privilege: Public

Associations:

The InSessionEcsRPC_S class has associations with the following classes:
CsGateWay (Aggregation)

4.3.58 InSessionExtRPC_C Class

Parent Class: Not Applicable
Public: Yes
Distributed Object: Yes
Purpose and Description:
This is the client/proxy implementation that defines the RPC (Remote Procedure Call) for delivering data message from the external Client to ECS/Ingest.

Attributes:

None

Operations:

extDAN - This is the RPC that delivers the Data Availability Notice (DAN) data message from the external client to ECS Ingest.
Arguments: handle_t InSessBH, char *DANmsg, char **DAAmsg, error_status_t *extDANstatus

Return Type: Void
Privilege: Public

extDDA - The is the RPC that delivers the Data Delivery Ack (DDA) from the external client to ECS Ingest.
Arguments: handle_t InSessBH, char *DDAmsg, error_status_t *status
Return Type: Void
Privilege: Public

Associations:

The InSessionExtRPC_C class has associations with the following classes:
Class: CsGateWay Invokes - The Gateway object interfaces with the InSessionExtRPC object to deliver the DAN and DDA data messages received from the external Client to Ingest Session.
InSession (Aggregation)

4.3.59 InSessionExtRPC_S Class

Parent Class: Not Applicable
Public: No
Distributed Object: Yes
Purpose and Description:
This is the server implementation that defines services for sending data messages from the external client to ECS Ingest.

Attributes:

None

Operations:

extDAN - This is the RPC that delivers the Data Availability Notice (DAN) data message to ECS Ingest.
Arguments: handle_t InSessBH, char *DANmsg, char **DAAmsg, error_Status_t *extDANstatus
Return Type: Void
Privilege: Public

extDDA - This is the RPC that delivers the Data Delivery Ack (DDA) from the external

client to ECS Ingest.
Arguments: handle_t InSessBH, char *DDAmsg, error_status_t *extDDAstatus
Return Type: Void
Privilege: Public

Associations:

The InSessionExtRPC_S class has associations with the following classes:
InSession (Aggregation)

4.3.60 InSessionInfo Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class: True
Purpose and Description:
Keeps track of all the sessions running under the Ingest Server.

Attributes:

ClientID - The identifier of the external client.
Data Type: char *
Privilege: Private
Default Value:

SessionID - The identifier of the Ingest Session.
Data Type: int
Privilege: Private
Default Value:

Operations:

DeleteSession - Deletes a session from the Ingest Server's session list.
Arguments: int SessionID
Return Type: int
Privilege: Public

ListSessions - Lists all sessions in the Ingest Server's session list.

Arguments: void
Return Type: int
Privilege: Public

SearchSession - Searches for a session in the Ingest Server's session list based on the session identifier.
Arguments: int SessionID
Return Type: int
Privilege: Public

SearchSession - Searches for a session in the Ingest Server's session list based on the external client identifier.
Arguments: char *ClientID
Return Type: int
Privilege: Public

Associations:

The InSessionInfo class has associations with the following classes:
Class: InServer IsManagedBy

4.3.61 InSessionIntRPC_C Class

Parent Class: Not Applicable
Public: Yes
Distributed Object: Yes
Purpose and Description:
This is the client/proxy implementation for exporting the data messages to the InSession object class.

Attributes:

None

Operations:

IntDDN - This is the RPC that exports the Data Delivery Notice (DDN) data message to the InSession object class.
Arguments: handle_t InSessBH, char *DDN, error_status_t IntDDNstatus
Return Type: Void

Privilege: Public

Associations:

The InSessionIntRPC_C class has associations with the following classes:
Class: InRequest Invokes - The InRequest object class interfaces with the InSessionIntRPC_C object class to export data message(s) to the InSession object class.
InSession (Aggregation)

4.3.62 InSessionIntRPC_S Class

Parent Class: Not Applicable
Public: No
Distributed Object: Yes
Purpose and Description:
This is the server implementation for exporting data messages to the InSession object class.

Attributes:

None

Operations:

InDDN - This is the RPC that exports the Data Delivery Notice (DDN) data message to the InSession object class.
Arguments: handle_t InSessBH, char *DDN, error_status_t InDDNstatus
Return Type: Void
Privilege: Public

Associations:

The InSessionIntRPC_S class has associations with the following classes:
InSession (Aggregation)

4.3.63 InShortDAA Class

Parent Class: InMessage
Public: No
Distributed Object: No
Purpose and Description:
This object class populates the short DAA (DAN Acknowledgement) data message to be sent to the external Client after the receipt of the DAN.

Attributes:

myShortDAA - This is the short DAA (DAN Acknowledgement) data message.
Data Type: ShortDAAmsg
Privilege: Private
Default Value:

Operations:

FillDAA - This function will package the short DAA for the DAN Acknowledgement.
Arguments: int DAAStatus, int DANSeqNo
Return Type: Void
Privilege: Public

Associations:

The InShortDAA class has associations with the following classes:
None

4.3.64 InShortDDN Class

Parent Class: InMessage
Public: No
Distributed Object: No
Purpose and Description:
This object class populates the short DDN (Data Delivery Notice) data message to be sent to the external Client after the data is archived.

Attributes:

myShortDDN - Short Data Delivery Notice (DDN) data message.
Data Type: Short DDN msg
Privilege: Private
Default Value:

Operations:

FillDDN - Populates the short DDN data message with the given status information.
Arguments: int DDNStatus, int DANSeqNo
Return Type: Void
Privilege: Public

Associations:

The InShortDDN class has associations with the following classes:
None

4.3.65 InSourceMCF Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class: True
Purpose and Description:
This class retains configuration information on source input files (i.e., source parameter name, parameter location). This class provides services to retrieve, delete, and add configuration information for a specific source metadata configuration.

Attributes:

myDataType - This attribute specifies the data type (e.g,, CER00, LIS00) associated with the source metadata configuration file.
Data Type: char*
Privilege: Private
Default Value:

myFileType - This attribute specifies the file type (e.g. metadata, science) associated with

the source metadata configuration file.
Data Type: char*
Privilege: Private
Default Value:

myVersionNumber - This attribute identifies the version number of the source configuration file.
Data Type: int
Privilege: Private
Default Value:

Operations:

AddParInfo - This service provides the ability to add a record to the existing InSourceMCF
Arguments:

DeleteParInfo - This service provides the ability to delete a record from the InSourceMCF
Arguments:

GetParInfo - This service returns information for a specific target parameter such as the source parameter name, parameter location, and computer data type
Arguments:

InSourceMCF - This is the constructor service.
Arguments:

Associations:

The InSourceMCF class has associations with the following classes:
Class: InMetadata defines - The InSourceMCF class defines the InMetadata class by providing format informatin on input metadata files.
Class: InTemplateEditor maintains - The InTemplateEditor class maintains the InSourceMCF class by providing the ability to create new instances and modify existing instances of the InSourceMCF class.

4.3.66 InStatusMonitor Class

Parent Class: InGUISession
Public: No
Distributed Object: No
Persistent Class:
Purpose and Description:
Provides operations personnel the capability to monitor the state of ingest request(s) via the GUI interface. It allows operations personnel to specify all ingest requests or particular ones of interest for viewing. This is a derived object class from the InGUISession object class. It inherits all data and service members provided by the InGUISession object class.

Attributes:

myRequestCriteria - The criteria information provided by the operations personnel which is to be used for the ingest request searching.
Data Type: struct *
Privilege: Private
Default Value:

myRequestIdList - The list of ingest requests that satisfies the search criteria specified by the operations personnel for request status monitoring.
Data Type: int []
Privilege: Private
Default Value:

Operations:

ProcessRequest - Invokes appropriate services to get the state of ingest request(s). This service overloads the ProcessRequest() service defined in the InGUISession object class.
Arguments: struct *RequestCriteria, int RequestIDList[]
Return Type: Void
Privilege: Public

Associations:

The InStatusMonitor class has associations with the following classes:
Class: InRequestList QueriesRequest(s)From - The InStatusMontiro object interfaces with InRequestList to locate InReqeust(s) for status monitoring.

4.3.67 InSystemThreshold Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class: True
Purpose and Description:
Persistent storage of Ingest system-level thresholds that limit ingest request traffic and data volume.

Attributes:

myIngestRequestThreshold - Site-wide threshold for total allowed number of ongoing ingest requests.
Data Type: int
Privilege: Private
Default Value:

myIngestVolumeThreshold - Site-wide threshold for total data volume of all ongoing ingest requests.
Data Type: int
Privilege: Private
Default Value:

Operations:

GetIngestRequestThreshold - Get the site-wide threshold for the maximum number of requests allowd to be processed concurrently.
Arguments: void
Return Type: int
Privilege: Public

GetIngestVolumeThreshold - get the site-wide threshold for total number of data volume to be processed concurrently.
Arguments: void
Return Type: int
Privilege: Public

SetIngestRequestThreshold - Set the site-wide threshold for the maximum number of requests allowed to be processed concurrently.
Arguments: int NewRequestTheshold
Return Type: void

Privilege: Public

SetIngestVolumeThreshold - Set the site-wide threshold for total number of data volume allowed to be processed concurrently.
Arguments: int NewVolumeThreshold
Return Type: void
Privilege: Public

Associations:

The InSystemThreshold class has associations with the following classes:
InThreshold (Aggregation)

4.3.68 InTemplateEditor Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Purpose and Description:
Provides operations personnel with the capabilities to edit the templates associated with the Ingest Preprocessing via the GUI interface.

Attributes:

TemplateType - Indicates the template to which is to be edited.
Data Type: int
Privilege: Private
Default Value:

Operations:

ProcessRequest - Allows operations personnel to edit the specified template.
Arguments: int TemplateType
Return Type: int
Privilege: Public

Associations:

The InTemplateEditor class has associations with the following classes:

Class: InDataTypeTemplate maintains - The InTemplateEditor class maintains the InDataTypeTemplate class by providing the ability to add new instances of the InDataTypeTemplate class.

Class: InFileTypeTemplate maintains - The InTemplateEditor Class maintains the InFileTemplate class by creating new instances of the InFileTemplate class.

Class: InSourceMCF maintains - The InTemplateEditor class maintains the InSourceMCF class by providing the ability to create new instances and modify existing instances of the InSourceMCF class.

4.3.69 InThreshold Class

Parent Class: Not Applicable

Public: No

Distributed Object: No

Persistent Class: True

Purpose and Description:

This object tracks all of the thresholds for the ingest system.

Attributes:

None

Operations:

None

Associations:

The InThreshold class has associations with the following classes:

Class: InRequestManager ProvidesThresholdsfor - InRequestManager retries system and external data provider based thresholds for ingest processing.

4.3.70 InThresholdController Class

Parent Class: InGUISession

Public: No

Distributed Object: No

Persistent Class:

Purpose and Description:
Provides authorized operations personnel the capability to view or to set an ingest threshold via the GUI interface. The types of ingest thresholds consist of: 1) ingest request threshold--maximum requests allowed to be processed concurrently, 2) data volume threshold--maximum allowed data volume to be processed concurrently, and 3) transfer retry threshold--number of data transfer retry attempts when failure occurred. The object class is derived from the InGUISession object class. It interits all the data and service members provided by the InGUISession object class.

Attributes:

myNewThreshold - The new value for the specified threshold with which the operations personnel requested to replace.
Data Type: int
Privilege: Private
Default Value:

myThresholdType - Indicates the type of threshold that is to be replaced. The types of ingest thresholds consist of: 1) ingest request threshold--maximum request allowed to be processed concurrently, 2) data volume threshold--maximum data volume allowed to be processed concurrently, and 3) data transfer retry threshold--number of transfer retry attempts when failure occurred.
Data Type: int
Privilege: Private
Default Value:

Operations:

ProcessRequest - Invokes appropriate services to get/set the ingest threshold value. This service overloads the ProcessRequest() service defined in the InGUISession object class.
Arguments: int ThresholdType, int NewValue
Return Type: Void
Privilege: Public

Associations:

The InThresholdController class has associations with the following classes:
Class: InThreshold Maintains

4.3.71 InTransferredData Class

Parent Class: Not Applicable
Public: No
Distributed Object: No
Persistent Class:
Purpose and Description:
Performs services to instantiate ingested data files. Upon the creation of the InTransferredData object class, if the Delivery Record file is provided, the object class would contain the information for the grouping of the data type with the corresponding files. Otherwise the data type information is provided in the input request.

Attributes:

myDataLocation - Indicates the disk location where the ingest files reside (after data transmission).
Data Type: char*[]
Privilege: Private
Default Value:

myDataTypeIdList - The set of data types associated with the ingest files.
Data Type: char*[]
Privilege: Private
Default Value:

myFileIdList - The names of ingest files.
Data Type: char*[]
Privilege: Private
Default Value:

myFileVolumeList - The file sizes corresponding to the ingest files.
Data Type: int[]
Privilege: Private
Default Value:

Operations:

GetDtInfo - Provides the data type information for the associated ingest files.
Arguments: char *DataTypeId
Return Type: int
Privilege: Public

GetFileInfo - Provides information of the ingest file (e.g., file name, file size).
Arguments: char *DataId, char *DataLocation, int FileVolume
Return Type: int
Privilege: Public

Associations:
The InTransferredData class has associations with the following classes:
Class: InDataTransferTask Populates - The InDataTransfer object gets information on the data types and the files from the InTransferredData object.

4.4 Ingest CSCI Dynamic Model

Information is provided for the critical Ingest Subsystem scenarios. The scenarios are each in two parts--the set of scenario steps and an event trace diagram. The set of scenario steps provides a text description of the interactions between object classes in a scenario. The event trace diagram pictorially describes the interaction between object classes and external interfaces participating in a scenario.

4.4.1 Automated Network Ingest (Get) Scenario

ECS performs automated network ingest upon receipt of a Data Availability Notice (DAN) stimulus. The Landsat-7, SDPF, TSDIS, and SCF interfaces are candidates for use of this protocol. In this scenario, ECS will perform the data transfer (get) from the external location to the ECS system.

If the external data provider is not a DCE client, there will an ECS Gateway that will translate the TCP/IP socket service class received from the external data provider to the corresponding Remote Procedure Call (RPC) provided by the ECS Ingest. In the scenario diagram, the "External Data Provider Process" would be the Gateway if the external data provider does not have DCE.

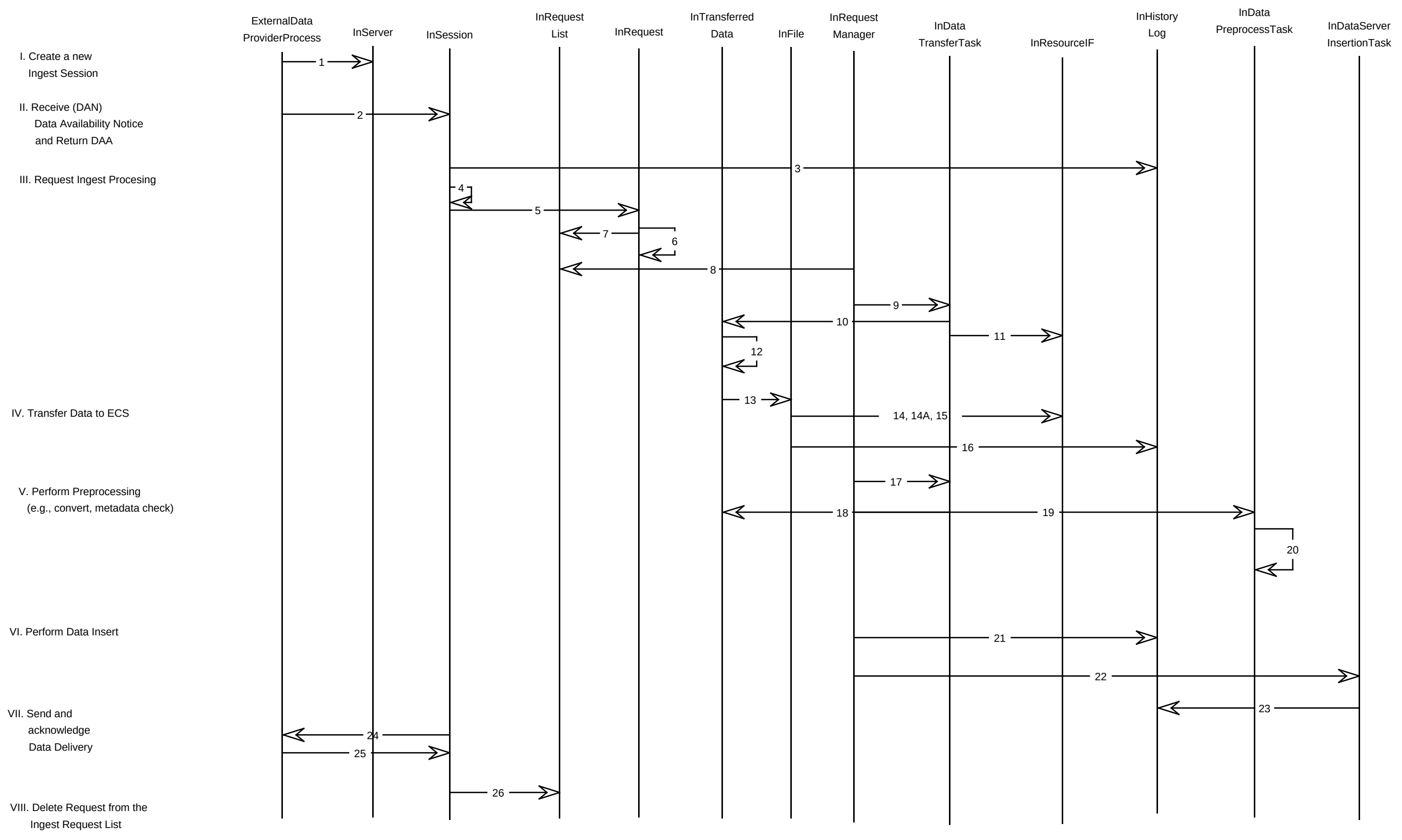
The following list describes the Automated Network Ingest (Get) scenario using object classes. Figure 4.4-1 is the corresponding event trace diagram. The numbers in the following list refer to the steps in the diagram:

Table 4.4-1. Automated Network Ingest Scenario (Get) Event Trace Diagram (1 of 2)

Step	Service	Description
1	CreateSession()	The Ingest server instantiates a new Ingest Session and setup the connection between Ingest Session and the External Data Provider process
2	extDAN()	External Data Provider process sends a DAN (Data Availability Notice) message to Ingest system. The DAN is verified and a DAA (DAN Ack) is returned.
3	WriteEvent()	Log the receipt of DAN
4	ProcessRequest()	Request to perform the automated network ingest request
5	constructor	Instantiate an InNetworkIngestRequest object
6	GetRequestId()	Assign an unique id for the ingest request
7	AddRequest()	Insert the ingest request onto the request list
8	GetNext()	Get the next request waiting to be processed
9	TransferDataByFile()	Begin the data transfer processing
10	constructor	Instantiate the InTransferredData object and populate it with the contents from the DAN message
11	AllocateResource()	Request allocation of an available staging device
12	GetInFileInfo()	Get the file information
13	constructor	Instantiate the InFile object
14	Transfer() data	Transfer file from external location to the resource object location in ECS Ingest system
14A	GetFileInfo()	Get the file information
15	Check() file	Check the existence and size of the transmitted file
16	WriteEvent()	Log the file transfer result
		Repeat steps 12-16 until all ingest files have been processed
17	GetDTInfo()	Request the Data Type information
18	GetDTInfo()	Get the Data Type information
19	constructor	Instantiate InDataPreProcessTask object
20	PreprocessData()	Performs appropriate data preprocessing (e.g., metadata check, conversion)

Table 4.4-1. Automated Network Ingest Scenario (Get) Event Trace Diagram (2 of 2)

Step	Service	Description
21	WriteEvent()	Log the sending of insert request to DataServer
22	SendInsert()	Send Insert Request to the appropriate DataServer
		See DataServer Event Trace for the actual insert detail
23	WriteEvent()	Log the completion status of data ingest
		Repeat steps 17-23 for each Data Type
24	ecsDDN	Send DDN (Data Delivery Notice) to the External Data Provider process informing the data transmission status
25	extDDA	The External Data Provider process returns DDA (Data Delivery Ack)
26	DeleteRequest()	Removes the ingest request from the Ingest Request List.



ExternalData ProviderProcess
InServer
InSession
InRequest List
InRequest
InTransferred Data
InFile
InRequest Manager
InData TransferTask
InResourceIF
InHistory Log
InData PreprocessTask
InDataServer InsertionTask
I. Create a new Ingest Session
II. Receive (DAN) Data Availability Notice and Return DAA
III. Request Ingest Procesing
IV. Transfer Data to ECS
V. Perform Preprocessing (e.g., convert, metadata check)
VI. Perform Data Insert
VII. Send and acknowledge Data Delivery
VIII. Delete Request from the Ingest Request List
1
2
3
4
5
6
7
8
9
10
11
12
13
14, 14A, 15
16
17
18
19
20
21
22
23
24
25
26

Figure 4.4-1. In_Automated_Network_Ingest_Get_Event_Trace Diagram Dynamic Model

4.4.2 Polling Ingest (Files) Scenario

The Polling Ingest (Files) scenario describes the mechanism by which the Ingest Subsystem acquires data from data centers which may not support an interprocess communication interface with ECS. The ECS/NESDIS and ECS/GDAO interfaces are candidates for use of this scenario. The following list describes the Polling Ingest (Files) Scenario. Figure 4.4-2 is the corresponding event trace diagram.

Table 4.4-2. Polling Ingest (Files) Event Trace Diagram (1 of 2)

Step	Service	Description
1	constructor	Instantiate polling session (start timer) to detect new files from the specified external directory
2	ProcessRequest()	Initiate ingest processing
3	constructor	Instantiate InDAN object class
4	GenerateDAN()	Generate a DAN file with file information retrieved from the specified directory location
5	constructor	Create Network Ingest Request Object using the generated DAN file as input
6	GetRequestId()	Assign an unique id for the polling request
7	WriteEvent()	Log receipt of Polling Request
8	AddRequest()	Add request to the list
9	GetNext()	Get the next request waiting to be processed
10	TransferDataByFile()	Begin data transfer on a file by file basis
11	constructor	Instantiate the InTransferredData object
12	AllocateResource()	Allocation for an available staging resource
13	GetFileInfo()	Get the file information
13A	Check()	Check the existence and size of the file
14	constructor	Instantiate the File object
15	Transfer()	Transfer file from Data Center to the resource object location in ECS Ingest system
16	GetFileInfo()	Check the existence and size of the transmitted file
17	WriteEvent()	Log file transfer result
		Repeat steps 13-17 until all files have been processed

Table 4.4-2. Polling Ingest (Files) Event Trace Diagram (2 of 2)

Step	Service	Description
18	GetDTInfo()	Request the Data Type Information
19	GetDTInfo()	Get the Data Type information
20	constructor	Instantiate InDataPreprocessTask object
21	PreprocessData()	Performs the appropriate data preprocessing (e.g., metadata check, conversion)
22	WriteEvent()	Log the sending of insert request to Data Server
23	SendInsert()	Send data insert request to Data Server
		See DataServer Event Trace for the actual insert detail.
24	WriteEvent()	Log the completion status of data ingest
25	CleanupDirectory()	Perform appropriate directory cleanup by means of moving the completed files to another directory
		Repeat steps 2-25 for ingesting next directory

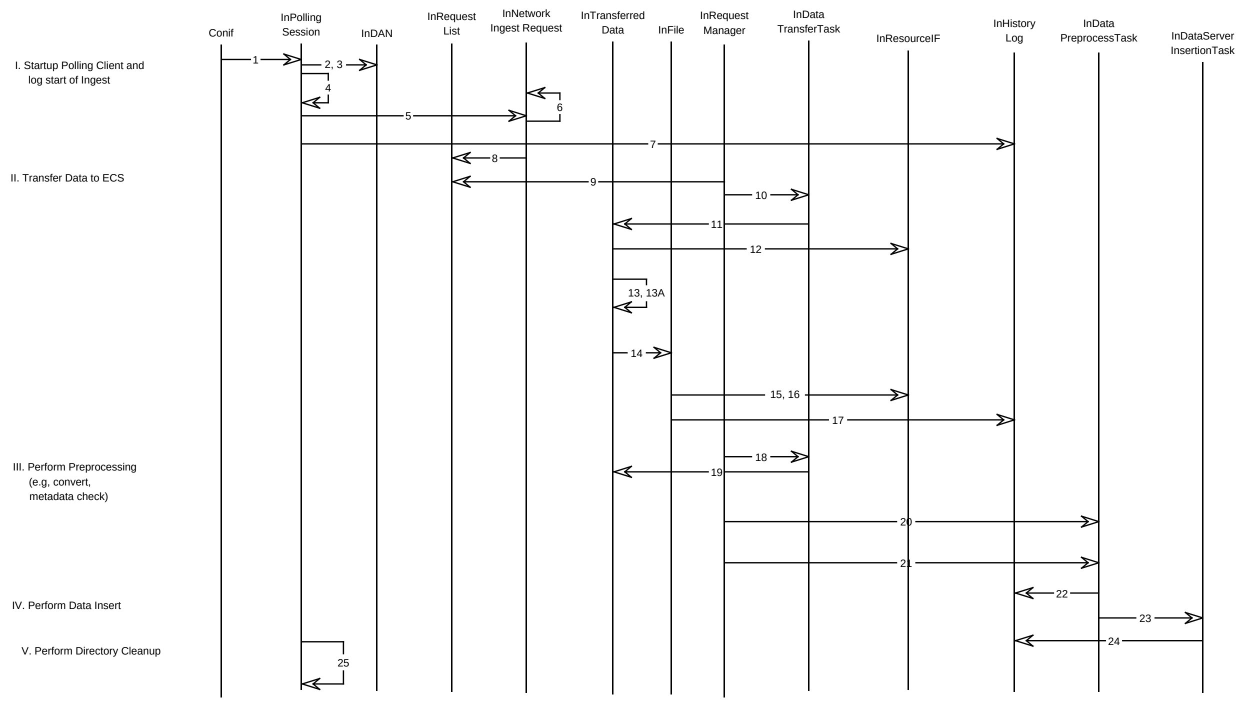


Figure 4.4-2. In_Polling_Files_Ingest_Event_Trace Diagram

4.4.3 Polling Ingest (Delivery Record) Scenario

The Polling Ingest (Delivery Record) scenario describes the mechanism by which the Ingest Subsystem acquires data from External I/Fs which control the initiation of data transfer. The EDOS interface is a candidate for use of the polling with delivery record scenario.

The following list describes the Polling Ingest (Delivery Record) Scenario. Figure 4.4-3 is the corresponding event trace diagram.

Table 4.4-3. Polling Ingest (Delivery Record) Event Trace Diagram (1 of 2)

Step	Service	Description
1	constructor	Instantiate polling session (start timer) to detect the Delivery Record file from the specified ECS directory
2	ProcessRequest()	Initiate Processing of request
3	constructor	Create Network Ingest Request Object using the Delivery Record File as input
4	GetRequestId()	Assign an unique id for the polling request
5	WriteEvent()	Log receipt of Polling Request
6	AddRequest()	Add request to the list
7	GetNext()	Get the next request waiting to be processed
8	TransferDataByFile()	Begin data transfer processing
9	constructor	Create the InTransferredData object
10	GetFileInfo()	Get the file information
11	constructor	Instantiate the File object
12	GetFileInfo()	Get the file information.
12A	Check()	Check the existence and size of the transmitted file
13	WriteEvent()	Log file verification result
		Repeat steps 10-13 until all files have been processed
14	GetDTInfo()	Request the Data Type Information from InDataTransferTask object.
15	GetDTInfo()	Get the Data Type information from InTransferredData object.
16	constructor	Instantiate InDataPreprocessTask object
17	PreprocessData()	Performs appropriate data preprocessing (e.g., metadata check, conversion)

Table 4.4-3. Polling Ingest (Delivery Record) Event Trace Diagram (2 of 2)

Step	Service	Description
18	WriteEvent()	Log the sending of insert request to Data Server
19	SendInsert()	Send data insert request to Data Server
		See DataServer Event Trace for the actual insert detail.
20	WriteEvent()	Log the completion status of data ingest
21	CleanupDirectory()	Perform appropriate directory cleanup by means of moving the completed files to another directory
		Repeat steps 14-21 for every Data Type

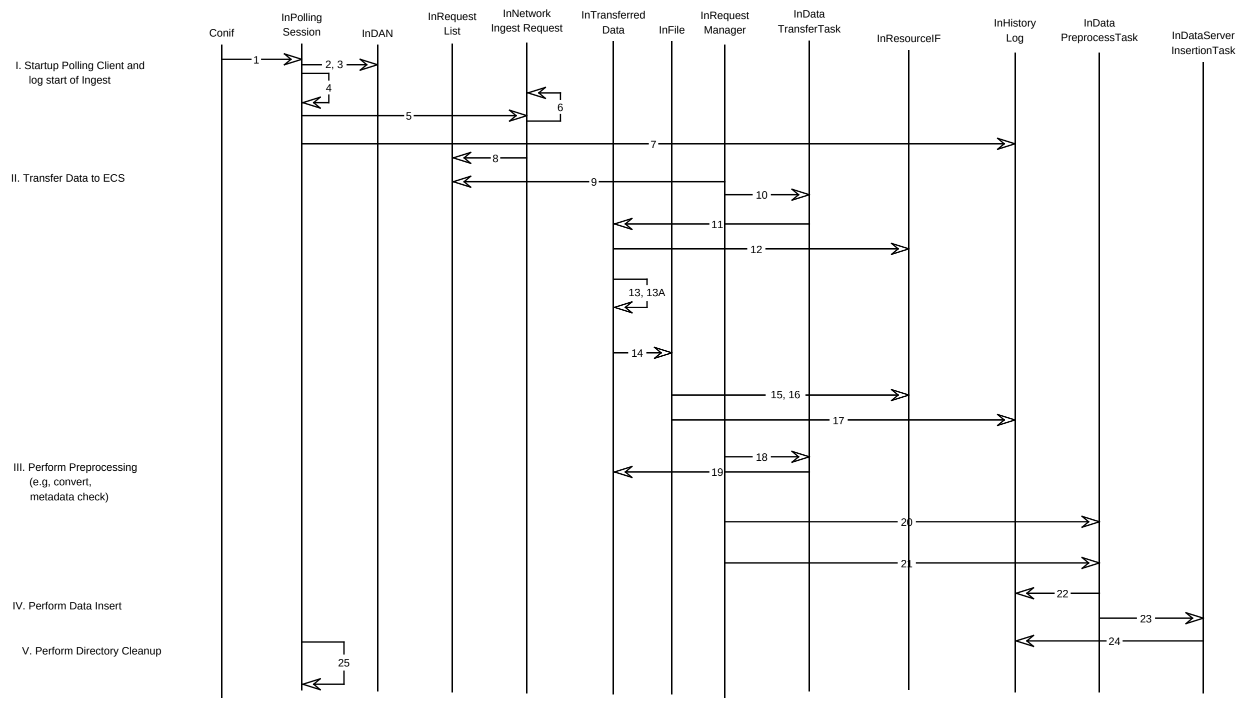


Figure 4.4-3. In_Polling_Delivery_Record_Ingest_Event_Trace Dynamic Model

4.4.4 User Network Ingest Scenario

The ECS provides the ECS users with the capability to perform interactive Network Ingest via the GUI Interface. An User Session will be configured on the User's GUI Interface to accept the request from the user via the GUI Interface and submit the request to the ECS system for data ingest.

The following list describes the User Network Ingest scenario using object classes. Figure 4.4-4 is the corresponding event trace diagram. The numbers in the following list refer to the steps in the diagram:

Table 4.4-4. User Network Ingest Event Trace

Step	Service	Description
1	N/A	User retrieves a Network Ingest Request Form via the WWW client
2	N/A	The Form is filled by the user and is submitted to the Ingest Form Script via HTTP daemon. (The HTTP daemon will parse the fields from the Form and invoke Ingest Form Script.)
3	N/A	The Ingest Form Script packages the fields from the Form into a DAN data message.
4	CreateSession()	The Ingest Form Script requests the ECS Ingest System for a new session to process the ingest request.
5	extDAN()	A DAN (Data Availability Notice) message is sent to the ECS Ingest system. The DAN is verified and a DAA (DAN Ack) is returned.
6	N/A	The Ingest Form Script reads the status from DAA and displays the status to the WWW client
		Refer steps 3 to 23 of Automated Network Ingest Scenario for detailed ingest processing occurred in the Ingest system.
7	ecsDDN()	When the ECS Ingest system completes the data ingest, it sends a DDN (Data Delivery Notice) to Ingest Form Script
8	extDDA()	The Ingest Form Script returns a DDA (DDN Ack) to the ECS Ingest system
9	N/A	The Ingest Form Script displays the ingest completion status to the WWW client

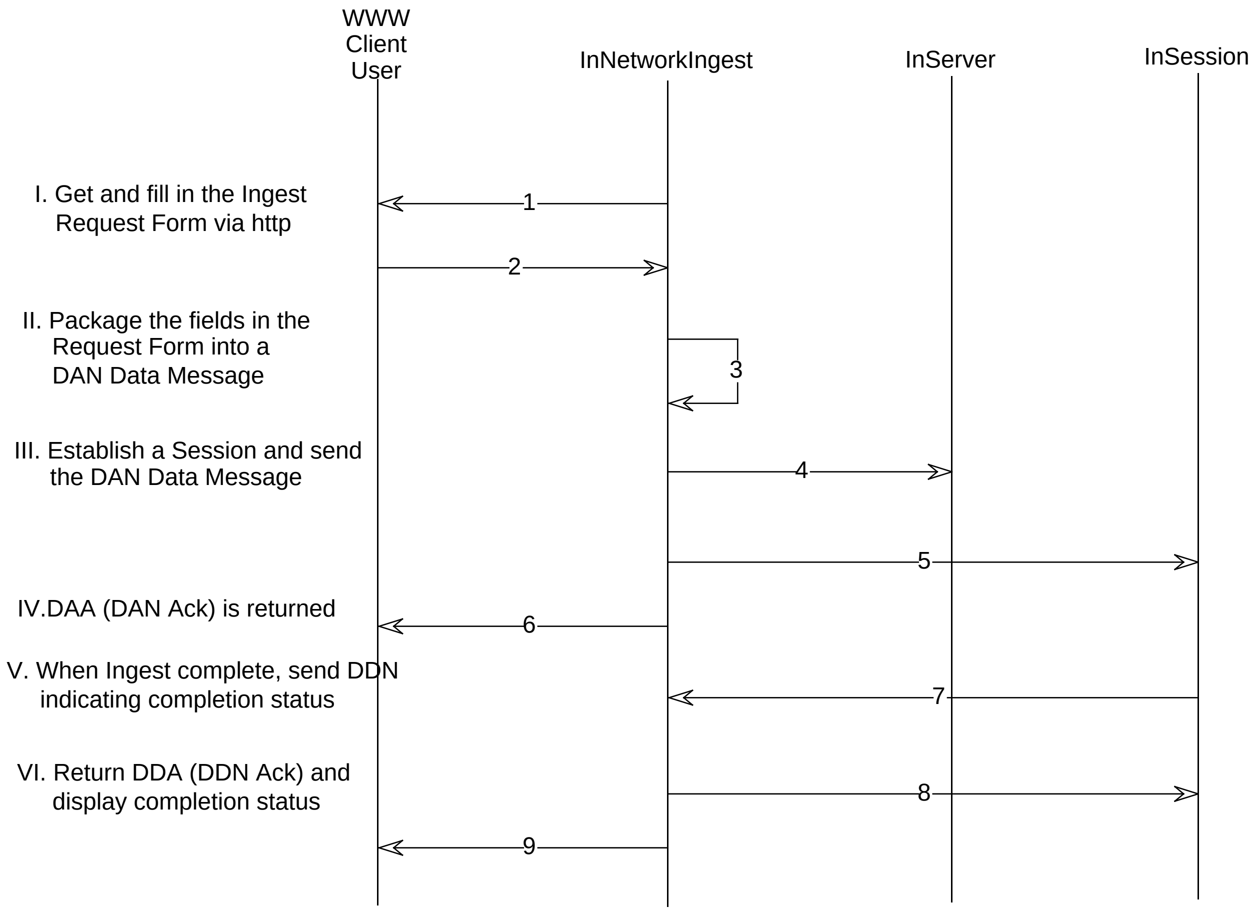

WWW Client User
InNetworkIngest
InServer
InSession
I. Get and fill in the Ingest Request Form via http
1
2
II. Package the fields in the Request Form into a DAN Data Message
3
III. Establish a Session and send the DAN Data Message
4
5
IV.DAA (DAN Ack) is returned
6
V. When Ingest complete, send DDN indicating completion status
7
VI. Return DDA (DDN Ack) and display completion status
8
9

Figure 4.4-4. In_User_Network_Ingest_Event_Trace Diagram Dynamic Model

4.4.5 Hard Media Ingest Scenario

The ECS system provides operations personnel with the capability to perform hard media (e.g., 8mm tape) ingest via the GUI Interface. A Media Ingest Session will be configured on the Operator's GUI Interface to accept the request from the Operator via the GUI Interface and submit the request to the ECS system for hard media ingest.

The following list describes the Hard Media Ingest scenario using object classes. Figure 4.4-5 is the corresponding event trace diagram. The numbers in the following list refer to steps in the diagram:

Table 4.4-5. Hard Media Ingest Event Trace

Step	Service	Description
1	constructor	Operator selects the Media Ingest option from the GUI interface and instantiates a media interface client to read inputs from the operator
2	CheckPrivilege()	The Media Ingest Session verifies Operator's privilege
3	ReceiveMsg()	Operator enters information needed for media ingest and the information is read in
4	ProcessRequest()	Request to perform the media ingest request
5	DownloadFiles()	Request to download files from the media to disk
6	AllocateResource()	Allocate an available staging device
7	AllocateResource()	Allocate an available peripheral device
8	CopyFile()	Copy files in bulk from the media to disk working area
9	DellocateResource()	Deallocate the staging device
10	DellocateResource()	Deallocate the peripheral device
11	CreateSession()	The Media Ingest Session requests the Ingest System for a new session to process the ingest request.
12	extDAN()	A DAN (Data Availability Notice) message is sent to the Ingest system. The DAN is verified and a DAA (DAN Ack) is returned.
13	SendMsg()	The Media Ingest Session reads the status from DAA and displays the status to Operator screen
		Refer to steps 2 to 20 of Polling Ingest (Delivery Record) Scenario for detailed ingest processing occurring in the Ingest system.
14	ecsDDN()	When the Ingest system completes the data ingest, it sends a DDN (Data Delivery Notice) to Media Ingest Session
15	extDDA()	The Media Ingest Session returns a DDA (DDN Ack) to the Ingest system
16	SendMsg()	And Displays the ingest completion status to the Operator screen

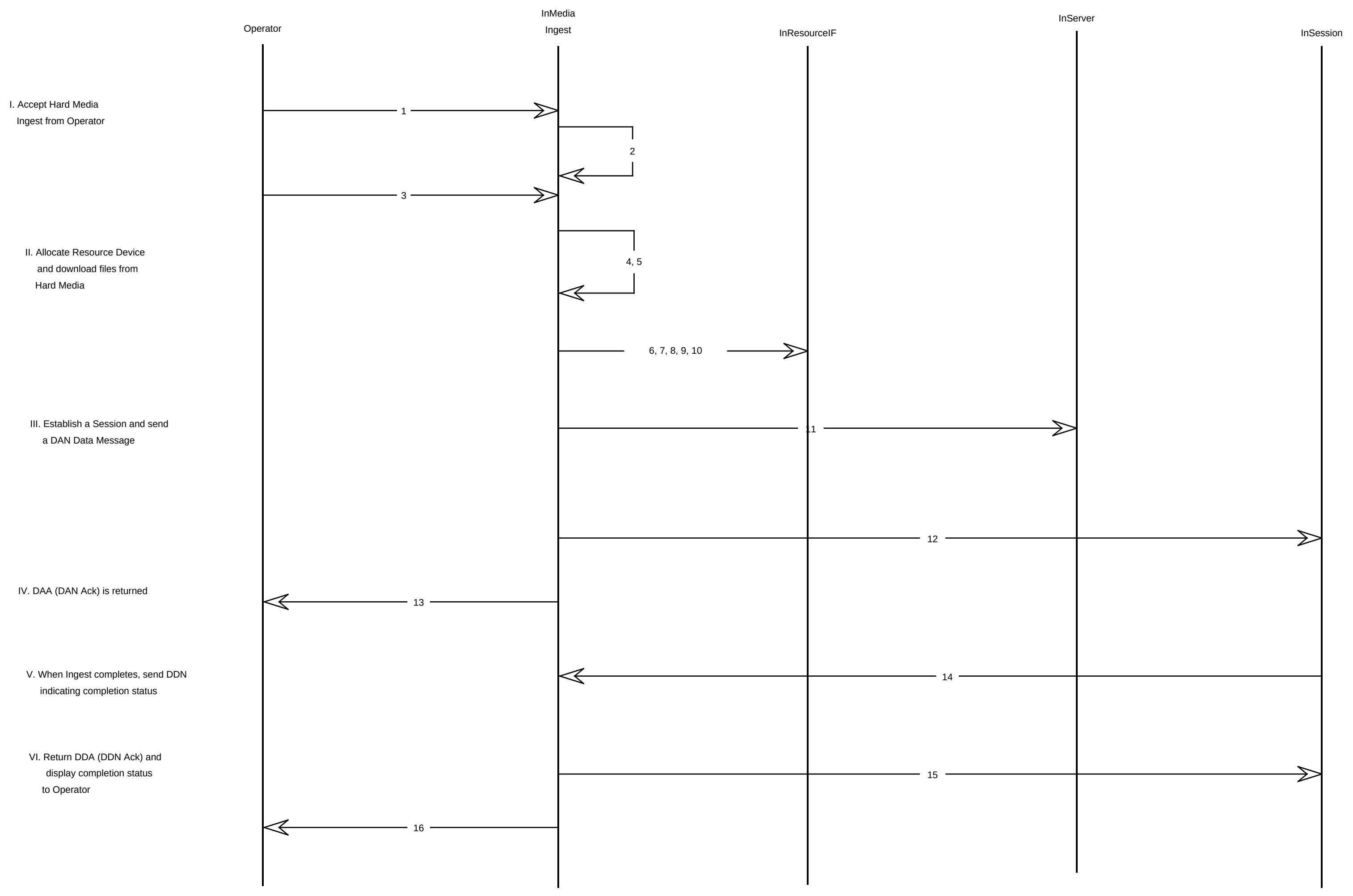

Operator
InMedia Ingest
InResourceIF
InServer
InSession
I. Accept Hard Media Ingest from Operator
II. Allocate Resource Device and download files from Hard Media
III. Establish a Session and send a DAN Data Message
IV. DAA (DAN Ack) is returned
V. When Ingest completes, send DDN indicating completion status
VI. Return DDA (DDN Ack) and display completion status to Operator
1
2
3
4, 5
6, 7, 8, 9, 10
11
12
13
14
15
16

Figure 4.4-5. In_Hard_Media_Ingest_Event_Trace Diagram

4.4.6 Ingest History Log Viewing Scenario

The ECS system provides operations personnel with the capability to view the Ingest History Log, a log that contains the results of all the past ECS ingest requests. The operations personnel have the capability to specify the search criteria (e.g., time range), the provider ID, data set name, and final request status of the Ingest History Log for log display. A LogMonitor screen will be configured on the Operator's GUI Interface to accept the log monitor request and the criteria specification from the operations personnel via the GUI Interface and invoke the appropriate service (provided by the Log object class) to get and display the Ingest History Log information to the operations personnel's GUI screen.

The following list describes the Ingest History Log Viewing scenario using object classes. Figure 4.4-6 is the corresponding event trace diagram. The numbers in the following list refer to the steps in the diagram:

Table 4.4-6. Ingest History Log Event Trace Diagram

Step	Service	Description
1	constructor	Operator selects the Ingest Log Viewing option from the GUI interface and instantiates an InLogMonitor object to read input from the operator
2	CheckPrivilege()	Verify the Operator's privilege for log viewing
3	ReceiveMsg()	Operator enters criteria information for the search and is read in as a log monitor request
4	ProcessRequest()	Request to perform log viewing
5	GetEvent()	Based on the criteria provided by the Operator, invoke the service to query the Ingest History Log according to the Operator's specification
6	DisplayResults()	Format and display the log results to the Operator's screen

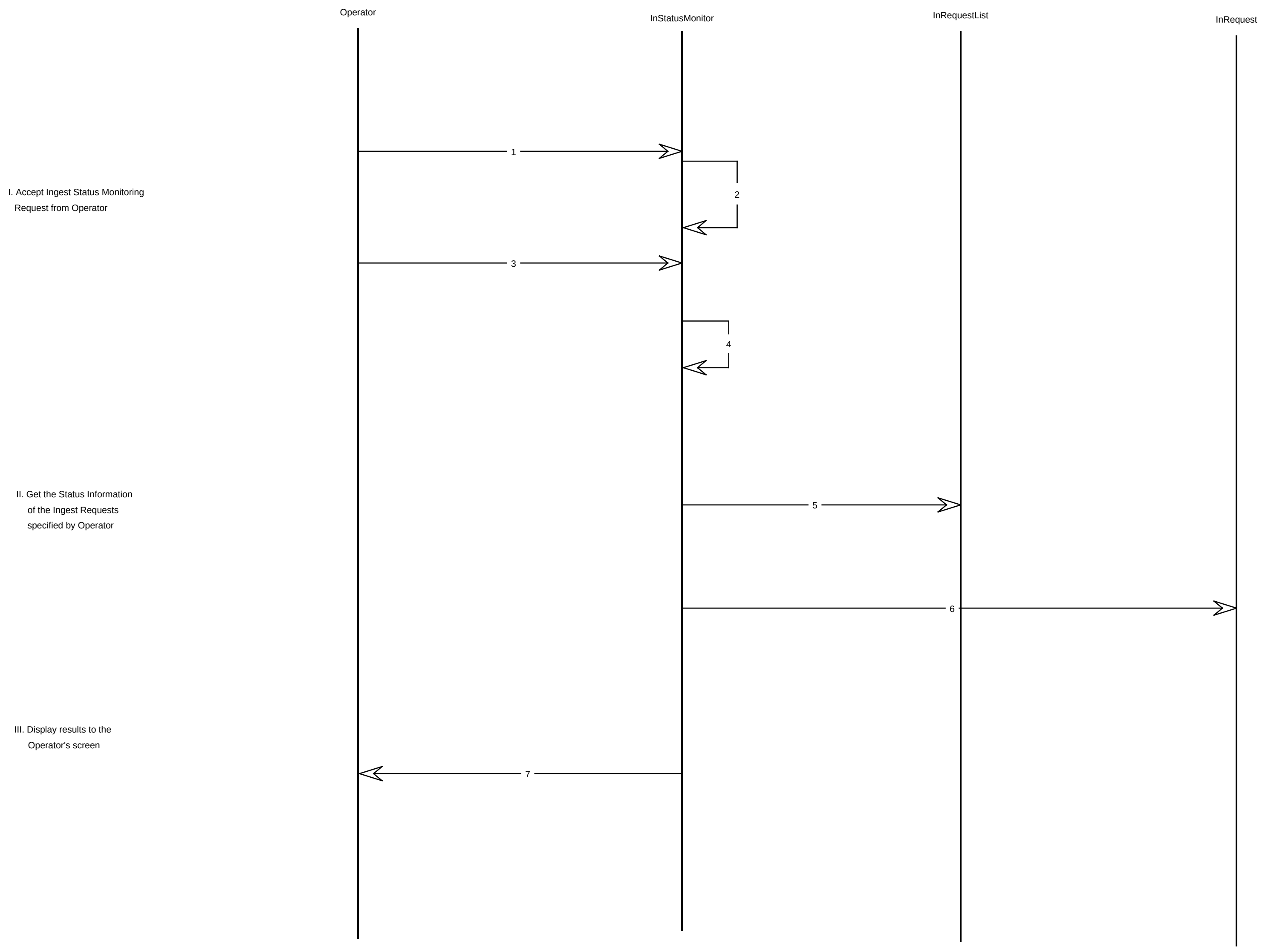

Operator
InStatusMonitor
InRequestList
InRequest
I. Accept Ingest Status Monitoring Request from Operator
1
2
3
4
II. Get the Status Information of the Ingest Requests specified by Operator
5
6
III. Display results to the Operator's screen
7

Figure 4.4-6. In_Ingest_History_Log_Viewing_Event_Trace Diagram Dynamic Model

4.4.7 Operator Ingest Status Monitoring Scenario

The ECS system provides operations personnel with the capability to monitor the status of the ingest requests that are in progress. The operator has the capability to look at the status of all requests or at only specific requests. An InStatusMonitoring screen will be configured on the Operator's GUI Interface to accept the ingest status monitoring request and the criteria specification from the operations personnel via the GUI Interface and invoke the appropriate service to get and display the request states information to the operations personnel's GUI screen.

The following list describes the Operator Ingest Status Monitoring scenario using object classes. Figure 4.4-7 is the corresponding event trace diagram. The numbers in the following list refer to the steps in the diagram:

Table 4.4-7. Operator Ingest Status Monitoring Event Trace

Step	Service	Description
1	constructor	Operator selects the Ingest Request Status Monitoring option from the GUI interface and instantiates a InStatusMonitor object to read input from the operator
2	CheckPrivilege()	Verify the Operator's privilege for request status viewing
3	ReceiveMsg()	Operator enters criteria information for the request search and is read in as a status monitor request
4	ProcessRequest()	Request to perform the request status monitoring
5	SearchRequest()	Based on the criteria provided by the Operator, search for the ingest request
6	GetStatus()	Get the status of the request
		Repeat step 6 for every request that satisfies the criteria specification
7	DisplayResults()	Format and display the states of ingest requests to the Operator's screen

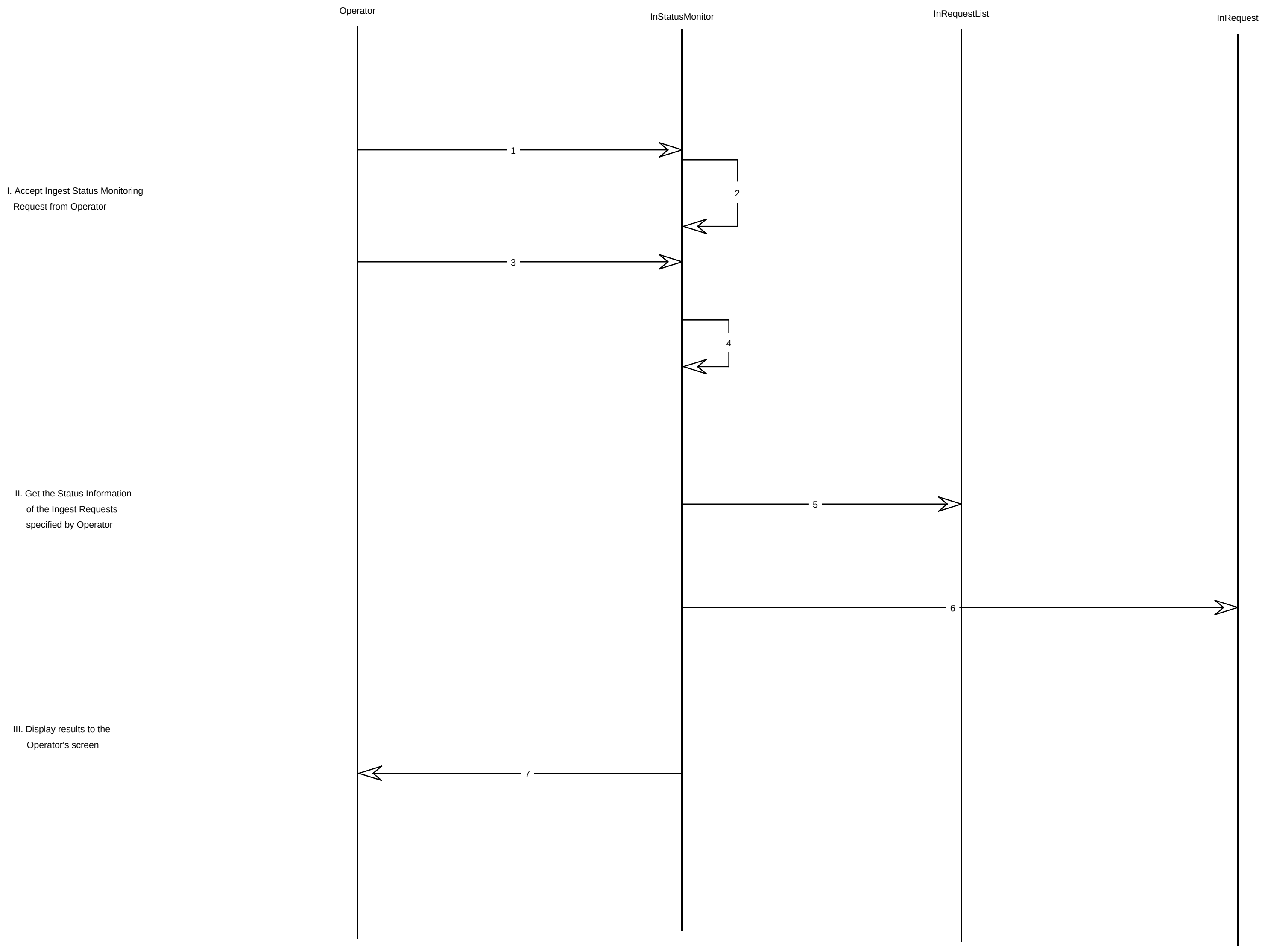

Operator
InStatusMonitor
InRequestList
InRequest
1
2
3
4
5
6
7
I. Accept Ingest Status Monitoring Request from Operator
II. Get the Status Information of the Ingest Requests specified by Operator
III. Display results to the Operator's screen

Figure 4.4-7. In_Ingest_Operator_Status_Monitoring_Event_Trace Diagram

4.4.8 User Ingest Status Monitoring Scenario

The ECS system provides the science users with the capability to monitor the status of the user's on-going ingest requests. A StatusMonitoring screen will be configured on the User's GUI Interface to accept the ingest status monitoring request from the user via the GUI Interface and invoke the appropriate service to get and display the request state information to the user's GUI screen.

The following list describes the User Ingest Status Monitoring scenario using object classes. Figure 4.4-8 is the corresponding event trace diagram. The numbers in the following list refer to the steps in the diagram:

Table 4.4-8. User Ingest Status Monitoring Event Trace

Step	Service	Description
1	constructor	User selects the Ingest Request Status Monitoring option from the GUI interface and instantiates a StatusMonitor object to read input from the user
2	CheckPrivilege()	Verify the User's privilege for status monitoring
3	ReceiveMsg()	User enters information needed for the request and is read in as a status monitor request
4	ProcessRequest()	Request to perform the user request status monitoring
5	SearchRequest()	Based on the criteria provided by the user, search for the ingest request
6	GetStatus()	Get the status of the request
		Repeat step 6 for every user owned request
7	DisplayResults()	Format and display the state of the user's requests to the User's screen

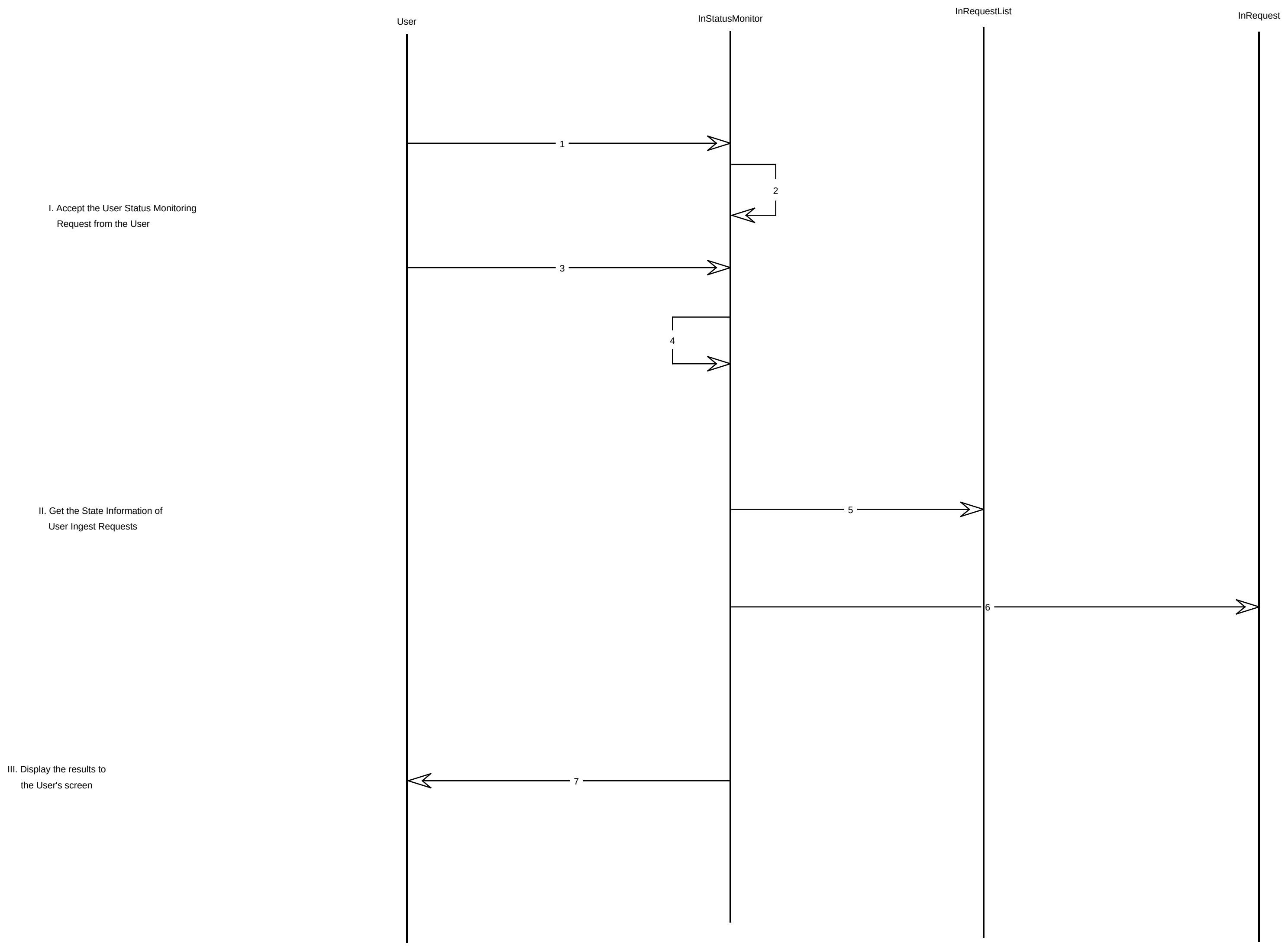

User
InStatusMonitor
InRequestList
InRequest
1
2
I. Accept the User Status Monitoring Request from the User
3
4
II. Get the State Information of User Ingest Requests
5
6
III. Display the results to the User's screen
7

_Figure 4.4-8. In_Ingest_User_Status_Monitoring_Event_Trace Diagram_

4.4.9 Operator Request Update Scenario

The ECS system provides operations personnel with the capability to update an on-going ingest request. The updates include: change priority, cancel, suspend, and resume (only the cancel option is available at Release A). An InRequestController screen will be configured on the Operator's GUI Interface to accept the update request from the operations personnel via the GUI Interface and invoke the appropriate service to perform the request update and display the update results to the operations personnel's GUI screen.

The following list describes the Operator Request Update scenario using object classes. Figure 4.4-9 is the corresponding event trace diagram. The numbers in the following list refer to the steps in the diagram:

Table 4.4-9. Operator Request Update Event Trace

Step	Service	Description
1	constructor	Operator selects the Request Update option from the GUI interface and instantiates an InRequestController object to read input from the operator
2	CheckPrivilege()	Verify the Operator's privilege
3	ReceiveMsg()	Operator enters request update information and is read in as a request update request
4	ProcessRequest()	Request to perform the request update
5	SearchRequest()	Based on the criteria provided by Operator, search for the ingest request
6	GetState()	Get the current state of the request
7	ChangePriority(), Cancel(), Suspend(), or Resume()	Invoke the appropriate service to perform the specified request update: Cancel(), Suspend, or Resume()
8	CancelTransfer(), SuspendTransfer(), or ResumeTransfer()	If the request is in Data Transferring state, ask the DataTransferTask object to perform the specified service: CancelTransfer(), SuspendTransfer(), or ResumeTransfer()
9	CancelPreprocess(), SuspendPreprocess(), or ResumePreprocess	If the request is in the Data Preprocessing state, ask the DataPreprocessTask object to perform the specified service: CancelPreprocess(), SuspendPreprocess(), or ResumePreprocess()
10	CancelInsert(), SuspendInsert(), or ResumeInsert()	If the request has already been sent to the Data Server for insertion, ask the DataServerInsertionTask object to perform the specified service: CancelInsert(), SuspendInsert(), or ResumeInsert()
11	SendMsg()	Display the request update results to the Operator's screen

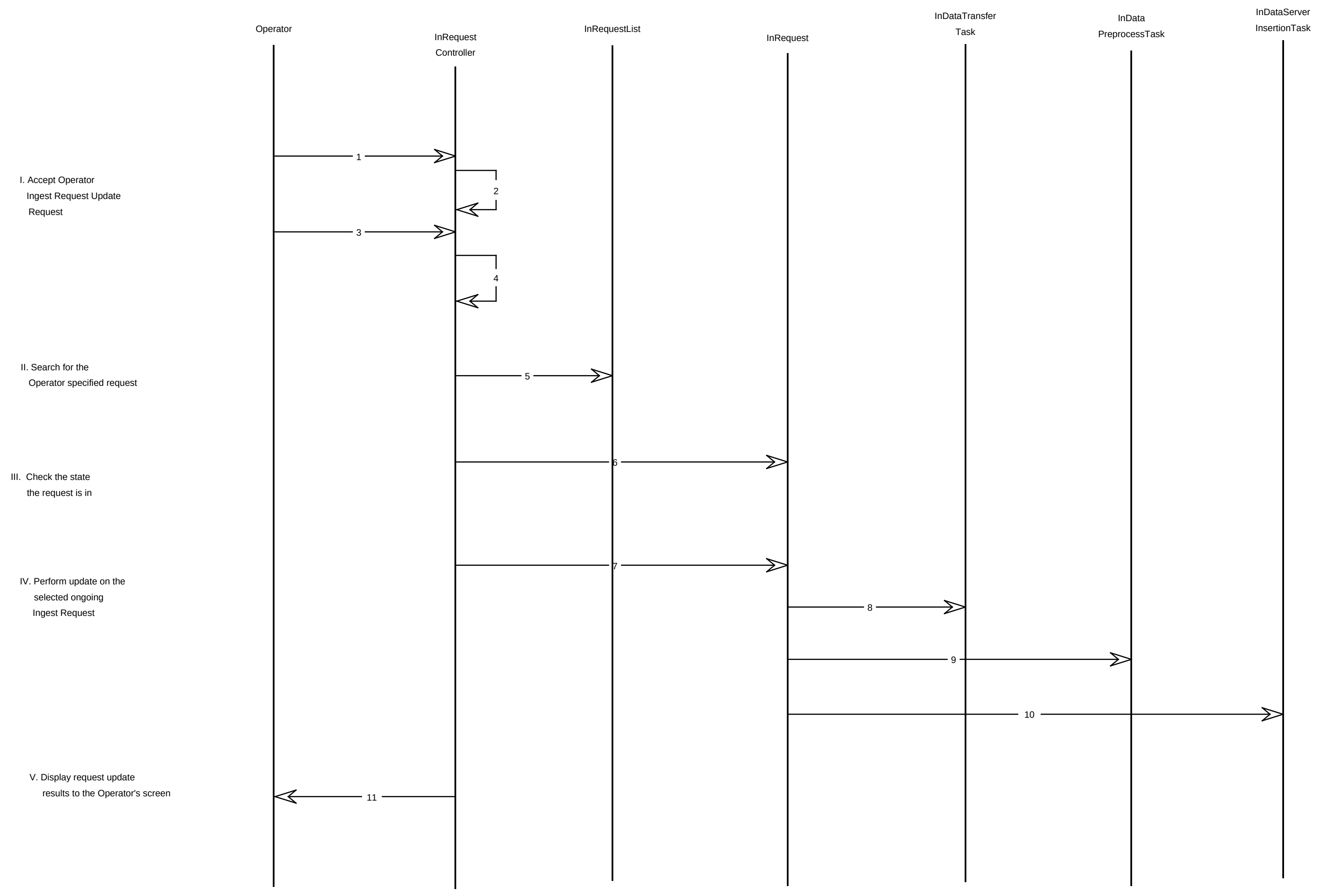

Operator
InRequest Controller
InRequestList
InRequest
InDataTransfer Task
InData PreprocessTask
InDataServer InsertionTask
I. Accept Operator Ingest Request Update Request
II. Search for the Operator specified request
III. Check the state the request is in
IV. Perform update on the selected ongoing Ingest Request
V. Display request update results to the Operator's screen
1
2
3
4
5
6
7
8
9
10
11

Figure 4.4-9. In_Ingest_Operator_Request_Update_Event_Trace Diagram

4.4.10 Preprocessing Scenario

The Ingest Subsystem provides services to preprocess all types of ingested data. Preprocessing includes extraction of metadata, conversion of metadata into a standard ECS format, metadata range/field checking, and converting/reformatting science and ancillary data.

The following scenario describes the interaction between the Preprocessing CSC and external classes from the initiation of data granule preprocessing to insertion of the preprocessed data into the Data Server Subsystem. The scenario applies to a data type granule which consists of a separate science and metadata file or a single data file where the metadata is embedded within the science data. Other categories of data type granules follow a similar scenario. Figure 4.4-10 is the corresponding event trace diagram.

Table 4.4-10. Preprocessing Event Trace (1 of 2)

Step	Service	Description
1	Preprocess	Initiate preprocessing task
2	InDataPreprocessList	Create an initial list of files to be inserted into the Data Server Subsystem
3	Preprocess	Initiate preprocessing on data granule
4	GetDTInfo	Obtain a list of file types associated with specific data type
5	GetNext	Get next file off of Input List created by Request Processing CSC
6	GetFileType	Obtain file type of file
7	GetFTInfo	Get information characterizing file type
8	InScienceData	Assume file type is science data, an instantiate correct InScienceData specialization
9	Preprocess	Execute required preprocessing on science data
10	InFile	Create a file object to store result of science data preprocessing
11	AddtoList	Add new file to Data Server insertion list
12	GetNext	Get next file off of Input List created by Request Processing CSC
13	GetFileType	Obtain file type of file
14	GetFTInfo	Get information characterizing file type
15	DsClDescriptor(GlClient &, UR &, DsSdTypeID &	Create appropriate DsClDescriptor object
16	GetMCF(ostream &)	Access target metadata configuration file
17	InFile	Create a file object to store target metadata configuration file

Table 4.4-10. Preprocessing Event Trace (2 of 2)

Step	Service	Description
18	InMetadata	Assume file type is metadata, and instantiate correct metadata specialization
19	Preprocess	Execute required preprocessing on metadata
20	PGS_MET_INIT	Initial metadata tool and load target metadata configuration file
21	PGS_MET_GetNext	Get next target metadata parameter
22	GetParInfo	Get information correlating target metadata parameter with source parameter name and location of required data
23	GetParVal	Perform necessary functions to obtain required value out of source metadata file
24	Read	Extract required value from source metadata file
25	PGS_MET_SET	Set value in target metadata configuration file
26	Repeat Steps 20-25	
27	PGS_MET_WriteFile	Write final target metadata configuration file into a PVL file
28	InFile	Create a file object store metadata PVL file
29	Validate	Validate metadata PVL file
30	AddtoList	Add new file to Data Server insertion list
31	SendInsert	Initiate data insertion
32	DsClESDTReferenceCollector(GlUR &dataserver, GlClient &, DsTSessionID=NULL	Open a data server session
33	DsClCommand(Advertisement&,GlParameterList &)	Create a command
34	SetParameters(pl: GlParameterList &)	Define files to be inserted
35	DsClRequest(cmd: DsClCommand*, pty: DsTRequestPriority, cat: DsTRequest(Category)	Create a request referencing associated commands
36	Submit(DsClESDTCollector&, GlURVector*=NULL):GlStatus	Submit Request

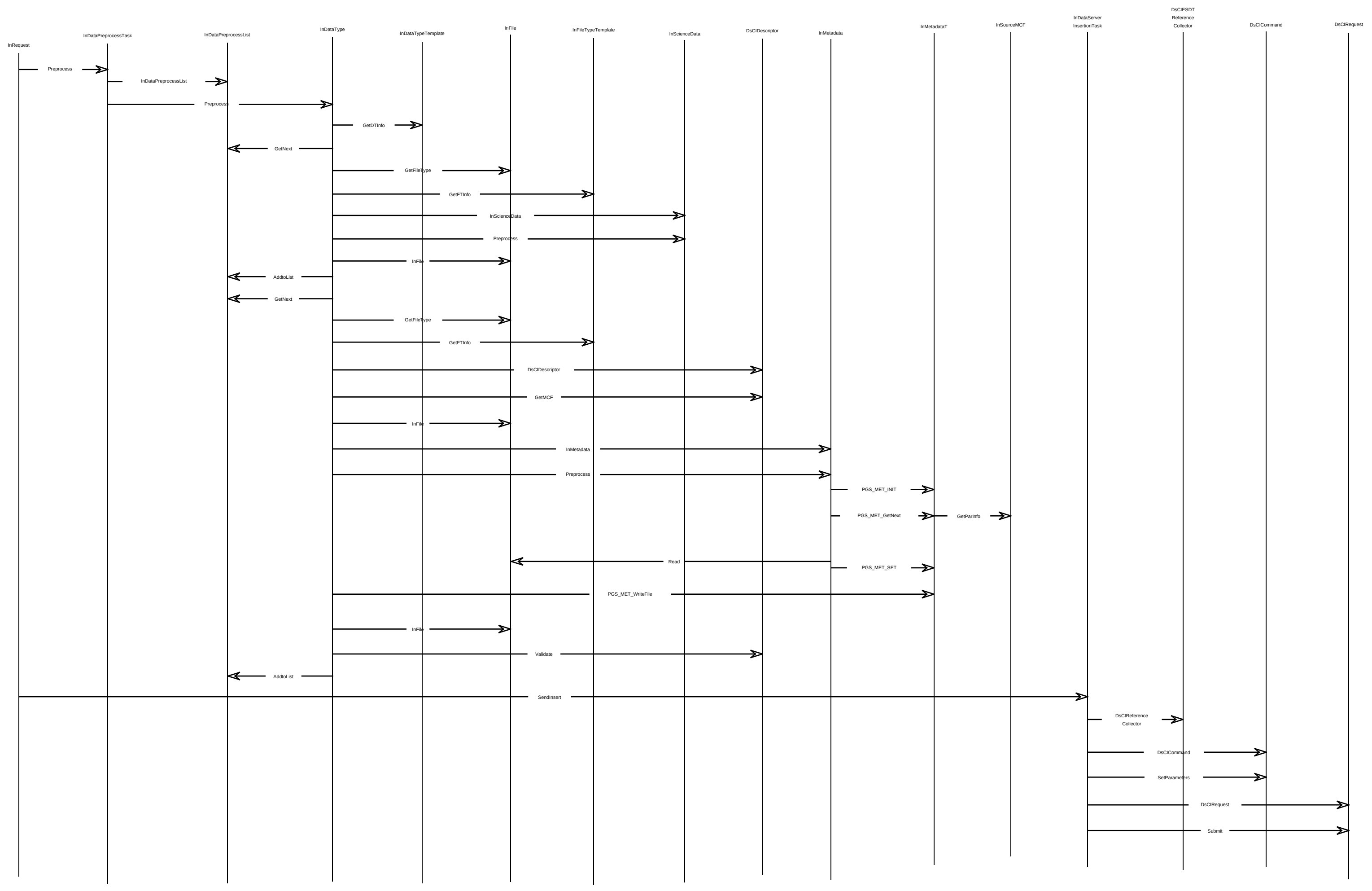


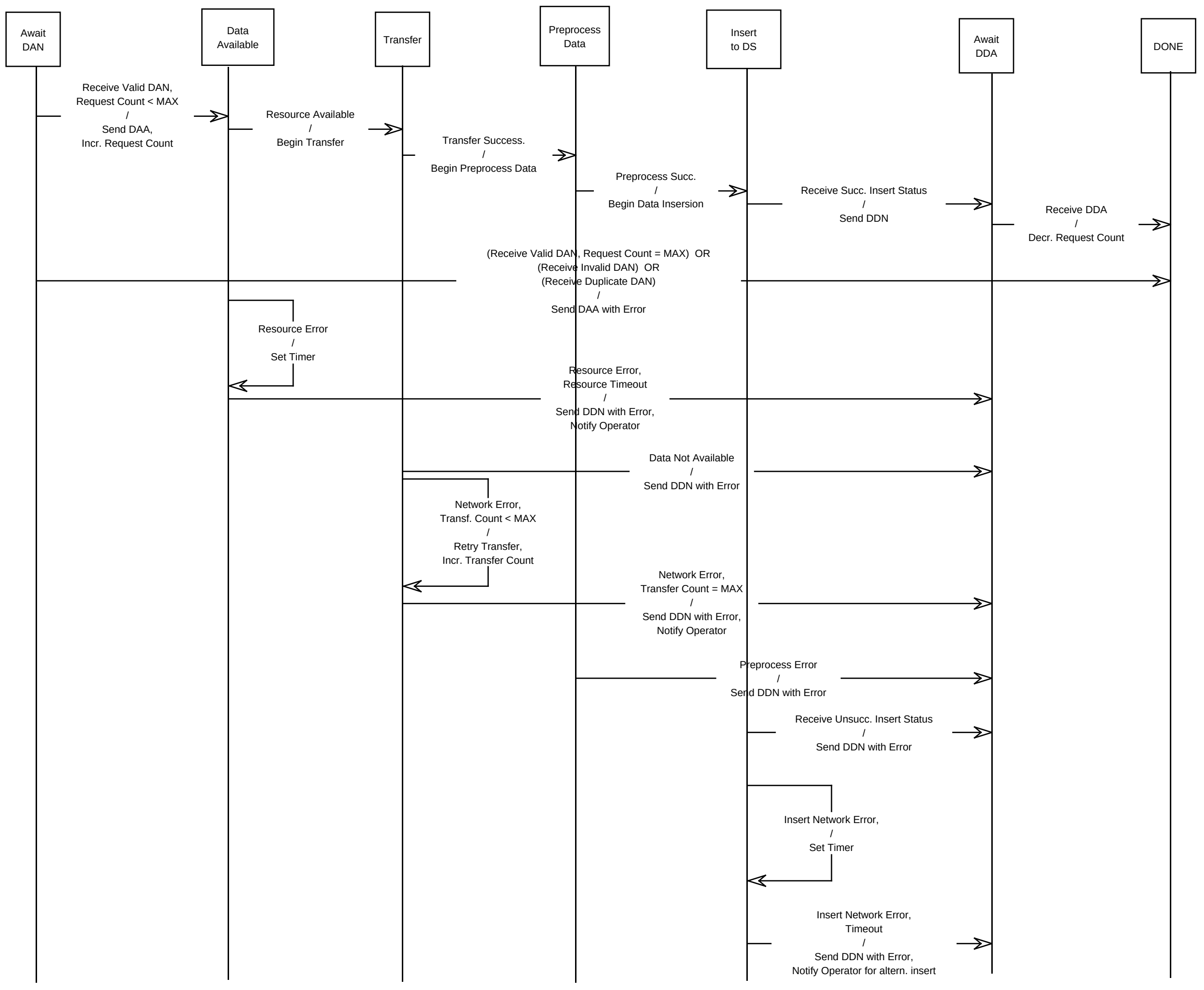
Figure4.4-10. In_Ingest_Preprocessing_Scenario1_Event_Trace Diagram

4.4.11 Ingest Fault/Error Scenario

This section presents the fault and error state machines for the ECS Ingest Subsystem. Two scenarios will be covered; one for the Ingest Processing and the other for the Media Handling processing.

Figure 4.4-11 illustrates the state machine for the whole Ingest processing function. Most faults and errors that occur during Ingest processing are related to the communications network. The identified fault and error categories are Transfer failure, Resource failure, and CI-to-CI interface failure. For each identified error category, an operator-tunable threshold will be defined indicating the number of retries the ECS Ingest software should attempt. Upon exhaustion of retry attempts (i.e., the retry threshold limit is met) and the error condition persists, the ECS Ingest software will alarm the ECS operations personnel. ECS operations personnel will perform diagnostics and try to fix the problem. If necessary, ECS operations personnel will coordinate with the external data source operations personnel to resolve the problem.

Figure 4.4-12 depicts the state machine for the Media Handling processing. Similar to the error categories identified for the communications network, the Resource failure is another error category where ECS operations personnel may be involved. Again after retry attempts have reached the threshold limit, the ECS Ingest software will alarm ECS operations personnel of the problem.



Await DAN
Data Available
Transfer
Preprocess Data
Insert to DS
Await DDA
DONE
Receive Valid DAN, Request Count < MAX / Send DAA, Incr. Request Count
Resource Available / Begin Transfer
Transfer Success. / Begin Preprocess Data
Preprocess Succ. / Begin Data Insersion
Receive Succ. Insert Status / Send DDN
Receive DDA / Decr. Request Count
(Receive Valid DAN, Request Count = MAX) OR (Receive Invalid DAN) OR (Receive Duplicate DAN) / Send DAA with Error
Resource Error / Set Timer
Resource Error, Resource Timeout / Send DDN with Error, Notify Operator
Data Not Available / Send DDN with Error
Network Error, Transf. Count < MAX / Retry Transfer, Incr. Transfer Count
Network Error, Transfer Count = MAX / Send DDN with Error, Notify Operator
Preprocess Error / Send DDN with Error
Receive Unsucc. Insert Status / Send DDN with Error
Insert Network Error, / Set Timer
Insert Network Error, Timeout / Send DDN with Error, Notify Operator for altern. insert

Figure 4.4-11. In_Ingest_Network_Ingest_Get_State_Diagram

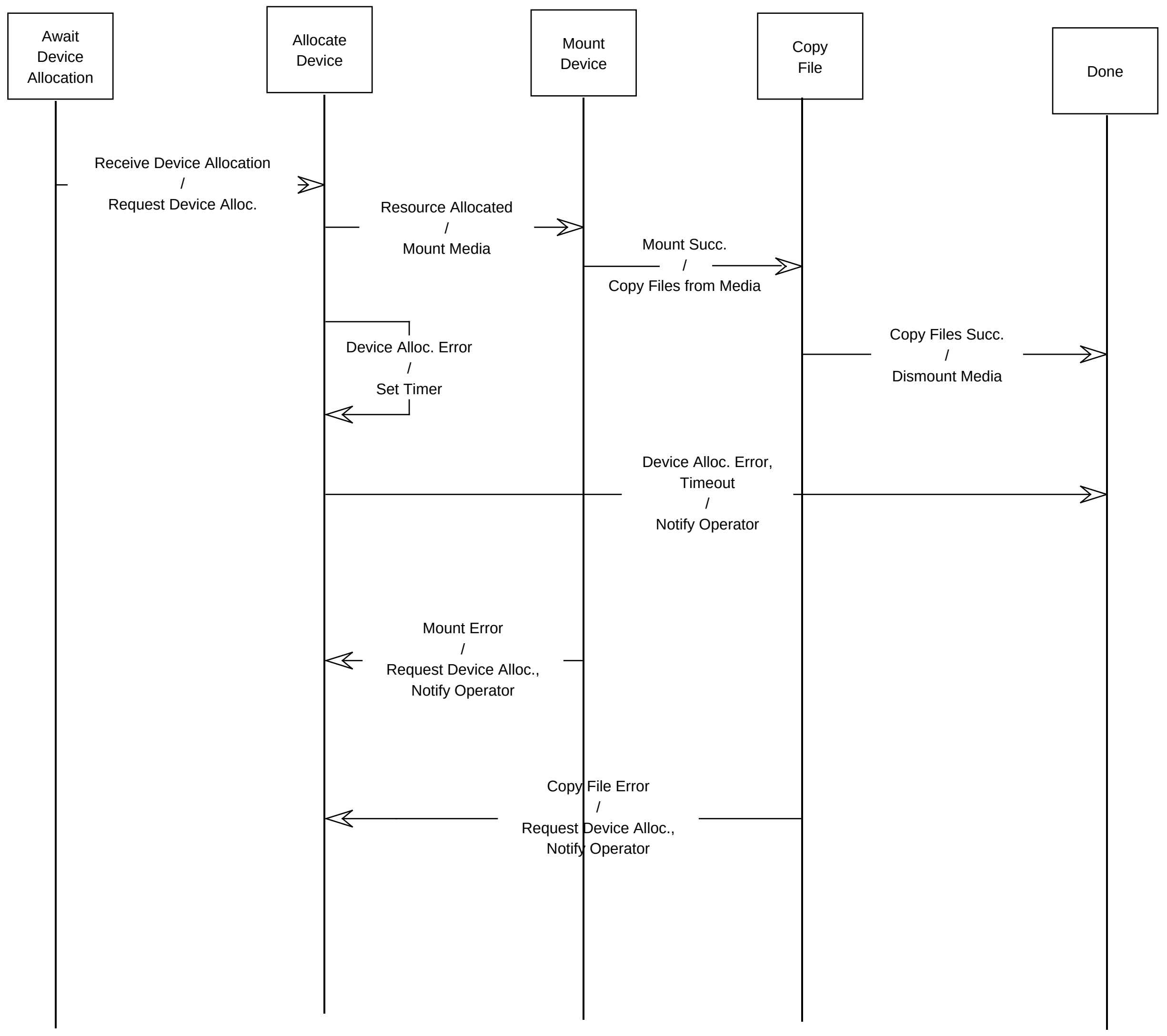

Await Device Allocation
Allocate Device
Mount Device
Copy File
Done
Receive Device Allocation / Request Device Alloc.
Resource Allocated / Mount Media
Mount Succ. / Copy Files from Media
Device Alloc. Error / Set Timer
Copy Files Succ. / Dismount Media
Device Alloc. Error, Timeout / Notify Operator
Mount Error / Request Device Alloc., Notify Operator
Copy File Error / Request Device Alloc., Notify Operator

Figure 4.4-12. In_Ingest_Media_Ingest_State Diagram

4.5 CSCI Structure

Table 4.5-1 shows the Computer System Components (CSCs) that comprise the Ingest CSCI. Details for each CSC are provided in the following paragraphs.

Table 4.5-1. Ingest CSCI Components

CSC	Description	Type (Custom=DEV; off-the-shelf=OTS)
Ingest Session Manager	Provides a template that instantiates other client CSCs.	DEV
Polling Ingest Client Interface	Polls for data files or Delivery Record files in an agreed location	DEV
Ingest Request Processing	Moderates ingest processing steps	DEV
Ingest Data Preprocessing	Performs required preprocessing and interface with the Data Server for data insertion	DEV
Ingest Data Transfer	Transfers data from source to ECS staging space	DEV
Operator Ingest Interface	GUI screens allowing operations staff ingest of hard media, on-going ingest request status monitoring, completed ingest request information viewing, ingest request controlling (e.g., canceling request), and ingest threshold controlling (i.e. to view or to set the threshold).	DEV
User Network Ingest Interface	GUI screens allowing users to ingest approved data and to perform ongoing ingest request status monitoring	DEV
Ingest DBMS	Data Base Management System used to store and provide access to the Ingest History Log and other ingest internal data	OTS
Ingest Administration Data	Provide services to access the History Log and administrative information of the Ingest Subsystem	DEV
Peripheral Software	Provide all media peripheral access software and operator administration functions for ingest peripherals	Reuse
Viewing Tools	Tools to allow displaying of ingested data for validation (analysis) purposes	Reuse
Data Storage Software	Software to store Level 0 data on working storage and repository storage (for one year)	Reuse
Resource Administration	Operator administration software to manage and control the Data Storage Software	Reuse
Client	Provides Science Data Server client interface services.	Reuse

Figure 4.5-1 shows the interactions of Ingest CSCs. Non-Ingest components are indicated by shading.

The Ingest Session Manager CSC sets up ingest sessions with external data providers (e.g., TSDIS and SDPF), via the CSS TCP/IP-to-OODCE Gateway and with other Ingest clients--Polling Ingest Client Interface CSC, User Network Ingest Interface CSC, and Operator Ingest Interface CSC. External data providers submit Data Availability Notices (DANs) to request data ingest. Polling Ingest Client Interface components poll accessible file system locations to detect data to be ingested; the component submits an equivalent DAN. The User Network Ingest Interface CSC allows an authorized science user to create and submit a DAN interactively. The Operator Ingest Interface CSC allows authorized operations staff to prepare physical media for ingest and to create an equivalent DAN.

The Ingest Session Manager CSC submits ingest requests (containing DAN data items) to the Ingest Request Processing CSC. The Ingest Request Processing CSC manages subsequent request processing. The Ingest Request Processing CSC invokes the Data Transfer CSC to transfer data from external locations. The Ingest Request Processing CSC invokes the Data Preprocessing CSC to preprocess ingested data (e.g., validate metadata parameters) and to insert data into the Data Server.

The shaded Data Server CSCs--Peripherals, Data Storage, Client, and Resource Administration--provide data storage and peripheral access services. In the case of Level 0 data ingest, the Data Server CSCs are reused in the Ingest Subsystem and implemented on Ingest Subsystem hardware. For non-Level 0 data, the Ingest and Data Server CSCs are implemented on Data Server hardware.

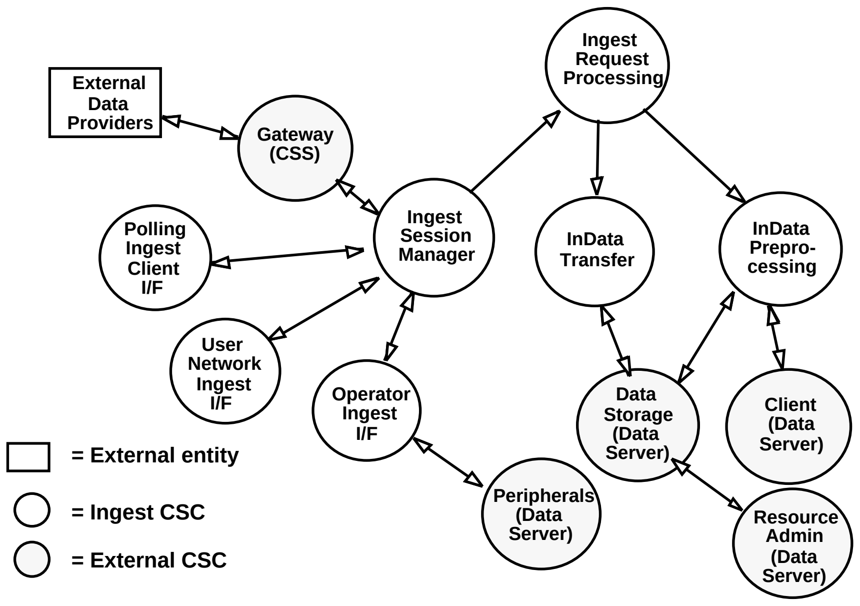


Figure 4.5-1. Ingest CSC Interaction

4.5.1 Ingest Session Manager CSC

The Session Manager CSC provides the fundamental capabilities to ingest data into the ECS system. The CSC can be tailored to fit in a specific interface and is comprised of the InServer, InServerExtRPC, InSession and InSessionExtRPC object classes. This CSC corresponds to the "Generic Ingest Client Shell" CSC described in the SDPS System Design Specification. Figure 4.5-2 illustrates the interaction between an external data source and the Ingest Session Manager CSC.

The InServerExtRPC and InSessionExtRPC object classes define the DCE Remote Procedure Calls (RPCs) interface. The client is responsible for invoking the RPCs to request Ingest services. The client initializes a connection to ECS Ingest Subsystem by invoking the CreateSession() RPC of the InServerExtRPC object class. This RPC creates an Ingest Session and establishes a connection with the external source. All the subsequent interactions with the external source are defined in the InSessionExtRPC object class.

If the external data source is not a DCE client, there will be an ECS Gateway that will receive the TCP/IP socket service calls from the external data source and translate the socket service calls into DCE RPC calls. The ECS Gateway will be supplied by the CSS Subsystem with support from Ingest. When the ECS Gateway receives an Authentication Request from the external source, it authenticates the client and invokes the CreateSession RPC of the InServerExtRPC. A session is created for the client and all other subsequent messages received over the TCP/IP sockets from that client are mapped to an RPC and forwarded to the associated InSessionExtRPC object class.

The InServer object class is instantiated from the main program Ingest Server. It sets up as a server and listens for incoming RPCs. It provides a single point of entry to the Ingest Subsystem for all ingest interfaces. When the external data source invokes the CreateSession() RPC, an Ingest Session is established and linked with the external data source by the Ingest Server. The InServer object class is responsible for managing all ingest sessions processing under the server (i.e., keeping track of sessions by adding the session to its list when a new session is created and deleting the session from its list when the session is terminated). A single session is created for a given client. Upon termination of the InSession, the InServer deletes the associated session and client information from its list. The InServer object class will be configured as a standalone program initiated at system startup on the Ingest Client HWCI.

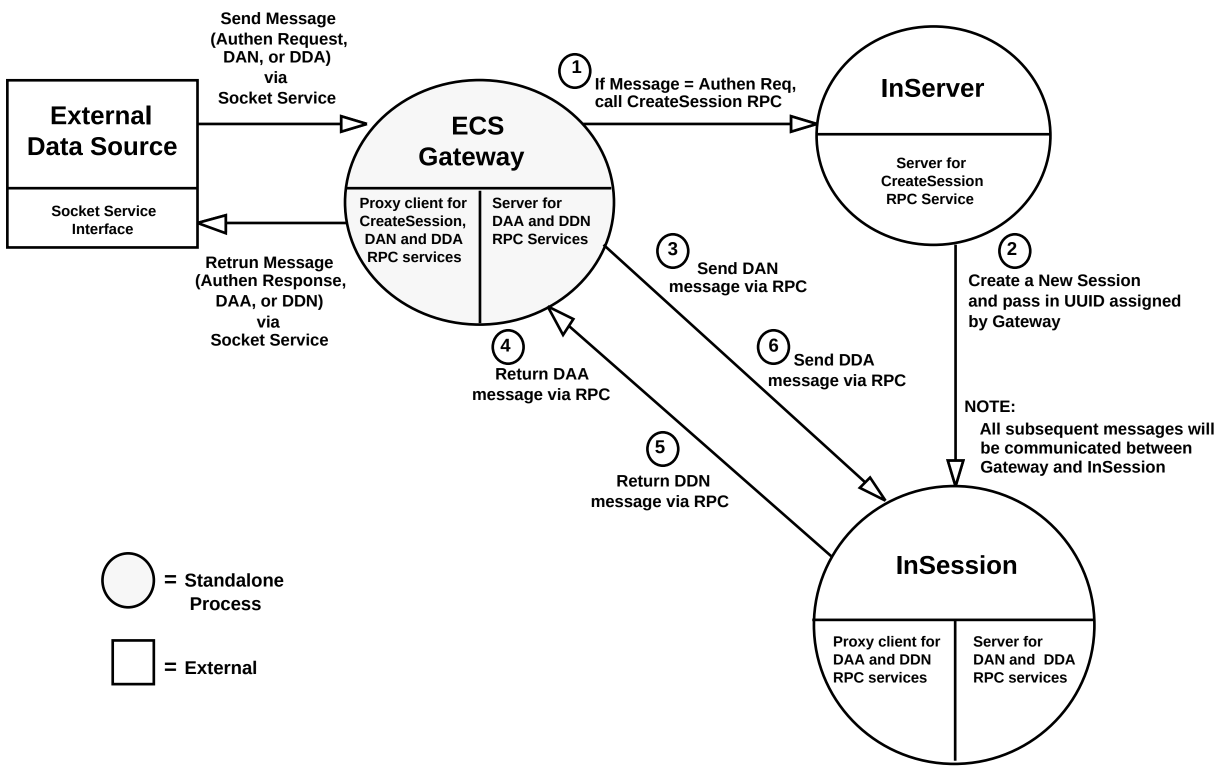
The InSession process is invoked by the CreateSession RPC of the InServerExtRPC object class. The InSession object class is instantiated from within the process, after which the session is setup as a server and listens for incoming RPCs from its client. The responsibility of the InSession object class consist of 1) managing multiple requests from a single client, where each request corresponds to a DAN message (RPC) with unique DAN sequence number, 2) creating the InRequest object class for each request and add to the InRequestList to be ingested (InRequest and InRequestList are object classes of the Ingest Request Processing CSC; refer to Ingest Request Processing CSC for detail), 3) cleaning up the request information upon receipt of the DDA message (RPC) , 4) sending the outgoing message to the client, and 5) terminating the connection with the client upon completion of all requests. In addition, the object class provides services to suspend, resume, and terminate a session. The InSession object class will be configured as a standalone program initiated by the InServer object class when the InServer receives the CreateSession RPC.

Table 4.5-2 shows the two object classes defined that are derived from the InSession object class; InPollingIngestSession and InGUISession. Refer to the corresponding CSC for additional details on the derived object classes:

Table 4.5-2. InSession Derived Object Classes

Derived Object Class	Corresponding Ingest Client CSC
InPollingIngestSession	Polling Ingest Session Manager
InGUISession	User Network Ingest Interface and Operator Interface

The Ingest Session Manager CSC supports the Automated Network Ingest Interface which provides external interface with the capability to ingest data in an automated fashion by means of network data transfer into the ECS system. The ingest process is initiated based on a stimulus provided by a DAN from the external interface. The TSDIS, Landsat-7, SDPF, and SCF interfaces are candidates for the use of the Automated Network Ingest Interface. This CSC corresponds to the "Ingest Clients for each External Interface" CSC described in the SDPS System Design Specification.Figure 4.5-2 shows the public services provided by the object classes of the Ingest Session Manager CSC.



Send Message (Authen Request, DAN, or DDA) via Socket Service
External Data Source
Socket Service Interface
Retrun Message (Authen Response, DAA, or DDN) via Socket Service
ECS Gateway
Proxy client for CreateSession, DAN and DDA RPC services
Server for DAA and DDN RPC Services
1
If Message = Authen Req, call CreateSession RPC
InServer
Server for CreateSession RPC Service
2
Create a New Session and pass in UUID assigned by Gateway
3
Send DAN message via RPC
4
Return DAA message via RPC
5
Return DDN message via RPC
6
Send DDA message via RPC
NOTE:
All subsequent messages will be communicated between Gateway and InSession
InSession
Proxy client for DAA and DDN RPC services
Server for DAN and DDA RPC services
= Standalone Process
= External

Figure 4.5-2. Ingest Session Manager CSC

4.5.2 Polling Ingest Client Interface CSC

The Polling Ingest Session CSC provides ECS with the capability to ingest data from data centers with little or no handshaking. The CSC is comprised of the InSession and InPollingIngestSession object classes. This CSC corresponds to the "Ingest Clients for each External Interface" CSC described in the SDPS System Design Specification. Figure 4.5-3 provides a pictorial overview of the Ingest Polling processing.

The responsibilities of the InSession Object Class consist of 1) creating the proper polling request, 2) detecting new files of interest at a tunable period of time in an external or local disk location, 3) creating the InRequest Object Class (a component of the Ingest Processing CSC (refer to Ingest Processing CSC for ingest details), 4) adding requests to the InRequestList Object Class, and 5) reporting the status of its ongoing ingest requests

InSession is the base object class of InPollingIngestSession; and therefore the InPollingIngestSession object class inherits all the data and service members from the InSession Object Class. The NESDIS, GSFC Data Assimilation Office (DAO), and EDOS interfaces are candidates for using the polling ingest protocol. The InPollingIngestSession (Files) Scenario addresses the NESDIS and DAO interface, while the InPollingIngestSession (Delivery Record) Scenario addresses the EDOS interface. The InPollingIngestSession (Delivery Record) Scenario proposes that the Ingest Subsystem detect and read information contained in a delivery record on a local ECS resource. The InPollingIngestSession (Files) Scenario proposes that the Ingest Subsystem detect and read information contained in a remote directory. The functionality to initiate both scenarios is contained in the Polling Ingest Session CSC.

4.5.3 Ingest Request Processing CSC

The Ingest Request Processing CSC is the core component of the Ingest Subsystem. It manages the ingest request traffic and the processing of the ingest requests. The CSC provides the capability to process multiple ingest requests concurrently. The CSC is responsible for tracking the ingest requests and coordinating the ingest processing, which is comprised of transferring data, performing data preprocessing, and sending an insertion request to the appropriate Data Server. The CSC is composed of InRequestList, InRequest, InRequest_C, InRequest_S, InRequestHeader, InRequestData, InRequestFile, InThreshold, InSystemThreshold, InExternalDataProvider, InRequestManager, InRequestManager_C, and InRequestManager_S object classes. This CSC is a new CSC not described in the SDPS System Design Specification.

The IngestRequestProcessing CSC follows the object factory model approach. In the factory model a "factory object" is established to create other objects based on a client request. The factory object provides a client and server component. For the Ingest Processing CSC the factory object is InRequestManager_S, which operates as a "server". The client proxy object is InRequestManager_C. It is linked with the "client". InRequestManager_S creates InRequest_S objects. The InRequest_S object handles all subsequent data preparation and insertion into the Data Server. Once InRequest_S objects are created, they communicated with their client counterparts, InRequest_C. Figure 4.5-4 shows the interaction of the generic object classes involved in the Ingest Request Processing CSC.

At system startup, the CSC is configured as a standalone program, with only InRequestManager_S instantiated, on the Ingest Client HWCI. The InRequest_C and InRequestManager_C objects are instantiated as client proxies (within the InSession object class or a subclass) when a new request is to be submitted. InRequest_C is invoked by the client (InSession) to create a new request. InRequest_C invokes the CreateRequest service of InRequestManager_C (step 1 in Figure 4.5-1). InRequestManager_C invokes an RPC call to InRequestManager_S (step 2), which creates InRequest_S in a separate pthread (step 3). One pthread is created for each InRequest_S object. The distributed object reference (OID) for InRequest_S is returned to the client proxy. InRequest_C handles subsequent communications with InRequest_S (step 4). In particular, InRequest_C invokes the SubmitRequest service of InRequest_S to trigger subsequent data transfer, preprocessing, and insert into the Data Server. InRequest_C waits for the return of status upon completion of the data insertion.

The client (InSession) checkpoints persistent context information about the OODCE connections established above. In the event of a failure of the client, the OODCE context information is used to reestablish the connection and receive return status.

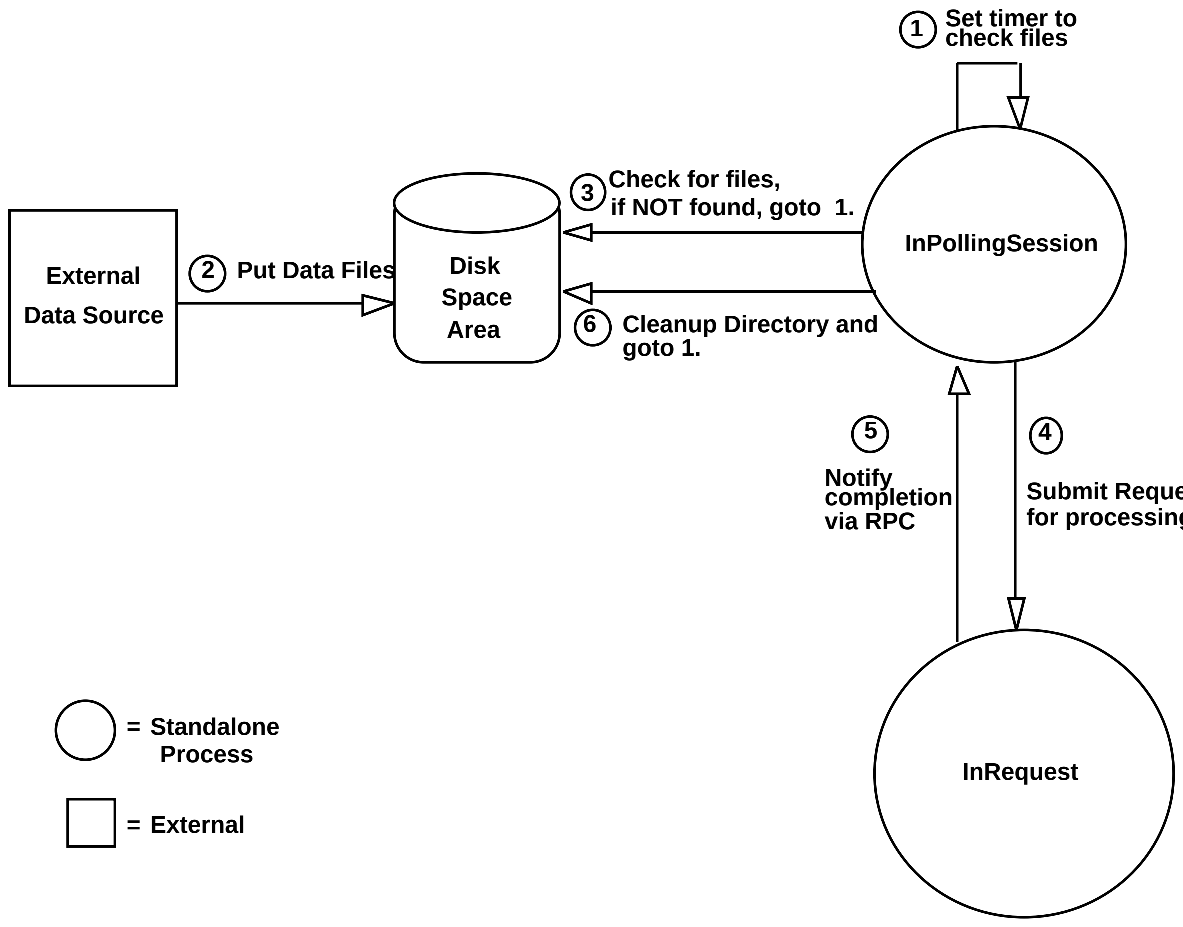

1 Set timer to check files
3 Check for files, if NOT found, goto 1.
External Data Source
2 Put Data Files
Disk Space Area
InPollingSession
6 Cleanup Directory and goto 1.
5 Notify completion via RPC
4 Submit Request for processing
InRequest
= Standalone Process
= External

Figure 4.5-3. Ingest Polling CSC

The InRequest_S object attributes (including OODCE context information) are checkpointed in a DBMS. Similarly, the InRequestList object contains a list of all the ingest requests that are currently ongoing or waiting to be processed. The InRequestList object attributes are checkpointed in a DBMS. In the event of a failure in InRequestManager_S or in InRequest_S, the InRequestManager_S object (after process restart, if needed) restores all ongoing requests using information checkpointed for InRequestList. Each InRequest_S object subsequently restores its contents, including OODCE context, from the checkpointed information.

The InRequest_S object contains information needed to start ingest processing for the request. The InRequest_S object invokes the Data Transfer and Data Preprocessing CSCs as libraries to perform ingest processing. The InRequest_S object stores the state (i.e., "active" or "completed") of the ingest request and the state (i.e., "not transferred", "transferred", "submitted to Data Server", and "Data Server insertion complete") of each data granule specified in the request. In addition, the InRequest_S object checks the integrity of the incoming ingest request by ensuring that all fields of the requests are properly filled. InRequest_S assigns an unique identifier (the distributed object reference) for the request.

4.5.4 Ingest Data Transfer CSC

The Ingest Data Transfer CSC provides services to the Ingest Client CSCI to facilitate transfer of data files into the Ingest Subsystem (on a file by file basis or bulk transfer), to collect information on individual ingested files, build lists of files to be ingested, and group files with valid data types. This collective set of services within the Ingest Data Transfer CSC is provided by the InDataTransferTask, InTransferredData, and InFile object classes. The CSC is configured as a standalone program initiated by the Ingest Request Processing CSC for every set of data granules to be transferred as supplied by the Ingest Request Processing CSC. This CSC is a new CSC not described in the SDPS System Design Specification.

The InDataTransferTask Object Class manages data transfer associated with a specific ingest request. The InDataTransferTask Class provides services to instantiate files, invoke data transfers (file by file or bulk), maintain retry thresholds, and allocate storage space via the InResourceIF Object Class.

The InTransferredData Object Class provides services to obtain file information on files associated with a specific transfer and to group a list of files with valid data types. This object class will provide services to the InDataTransferTask Object Class to obtain data type information.

The InResourceIF Object Class serves as the interface to the resource/device services by which the Data Server Subsystem provides. The Object class interfaces with the Storage Resource Management and the Data Distribution CSCIs of the Data Server Subsystem to perform these services. Refer to the Data Server volume for details.

The InFile Object Class provides services to instantiate ingested files on available storage space by collaborating with InResourceIFobject class services and perform necessary file checks.

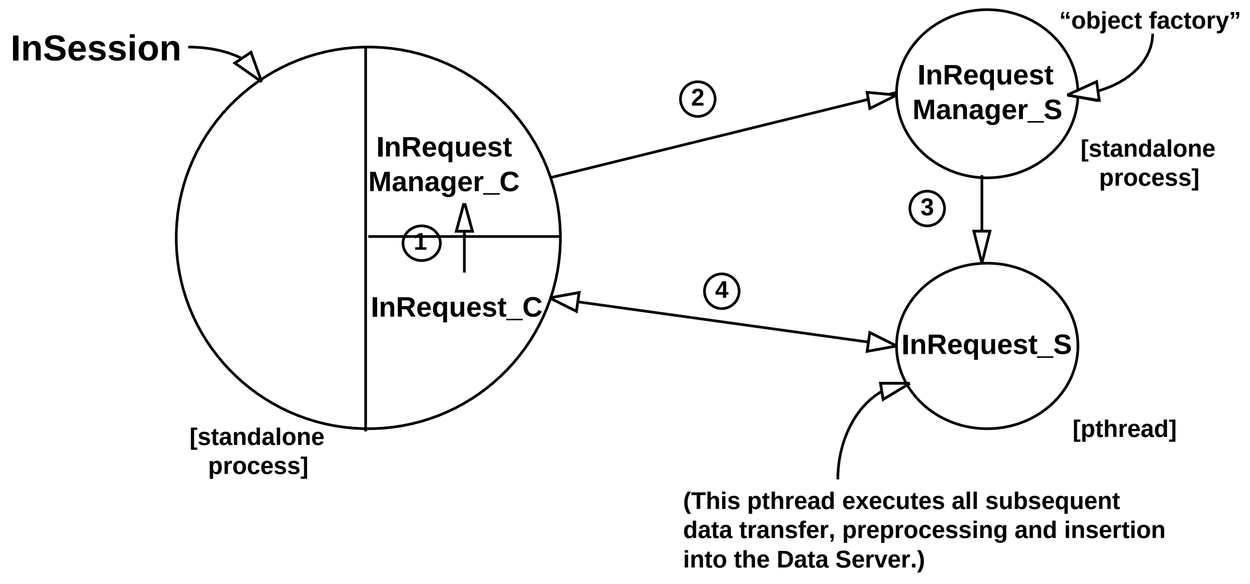


Figure 4.5-4. Ingest Request Processing CSC

4.5.5 Ingest Data Preprocessing CSC

The Ingest Data Preprocessing CSC provides services to perform required preprocessing of data and subsequent insertion of the data into the appropriate Data Server. The preprocessing of data consists of converting the data (if needed), extracting the metadata into the standard ECS metadata format (if needed), performing required metadata existence and parameter range checks, and updating the metadata with ingest specific metadata (e.g., start and stop date/time for ingest). In addition, the CSC is responsible for updating the request state, tracked by the Ingest Request Processing CSC, whenever its state changes; and accepting request cancellation, suspension, and resumption requests from the Ingest Request Processing CSC. This CSC corresponds to the "Translation Tools" and "Data Compression/ Decompression Tools" CSCs described in the SDPS System Design Specification.

The CSC depends on templates and configuration files to maintain data/file type policy and source/ target format information. These templates and configuration files supply necessary information to other classes within the CSC in order to perform the required preprocessing for a specific data type granule. The configuration files specify the input format and target format for specific data/ file types. Preprocessing classes rely on these files to specify how an ingested file is organized upon arrival (source metadata configuration file), and how the data should be organized before insertion (target metadata configuration file) into the data server. Most importantly, configuration files can be modified with no software recompilation or modification within the ingest preprocessing CSC. Source metadata configuration file modifications can be executed through an administration GUI. Target metadata configuration files are supplied by the Data Server Subsystem.

Each data type may consist of different file types (e.g., metadata, science, calibration). The constituents of each data type are defined in a data type template. This template has a record for each ingest data type specifying the file types that make up the data type. In addition, each file type has unique characteristics. For instance, each different file type uses different metadata configuration files. This type of information is contained in a file type template. These templates may be modified via an administration GUI. New entries (types) can be added to each of the templates without recompilation or modification within the preprocessing CSC.

The utilization of configuration files and templates and their interaction with preprocessing processes is illustrated in Figure 4.5-5, "Ingest Preprocessing CSC Data Flow". The preprocessing control library obtains information from the templates to determine which preprocessing tools are required (for a specific data/file type) and what the necesssary inputs are for the chosen tools (1). With the acquired information, preprocessing control can invoke the appropriate preprocessing tools (2). The configuration files aid the preprocessing tools in determining how the ingested input data is organized and the target format of the data (3). The preprocessing tools provide the functionality to obtain the values of required parameters directly from the ingested files (4). The preprocessing tools transform the ingested files into final products (5). Upon completion of preprocessing, a list of files is presented to the Data Server Subsystem for subsequent data archival (6).

The Ingest Preprocessing Object Model (presented in section 4.4) provides an object oriented view of the CSC. This object model presents preprocessing unique classes, as well as classes from other CSCs or subystems. Therefore, an understanding of the Preprocessing CSC and its interaction with external classes can be viewed in one continuous model. The Ingest Data Preprocessing CSC is composed of the InDataPreprocessTask, InDataType, InMetadata (and its subclasses), InScienceData (and its subclasses) , InDataTypeTemplate, InFileTypeTemplate, InSourceMCF, InDataPreprocessList, InMetadataTool, and InDataServerInsertionTask object classes. Each class is described in detail in the Preprocessing CSC data dictionary and their respective roles are illustrated in the detailed preprocessing scenario (Table 4.4-10). However, a general understanding of the object model can be gained by flowing through the following high level object-oriented scenario.

A "preprocessing task" is initiated when the InDataPreprocessTask Class is instantiated by the Ingest Request Processing CSC. As a result, the InDataPreprocessTask class invokes the preprocessing service of the InDataType Class to initiate preprocessing upon a set of files associated with a given data type (on a per granule basis). This set of files are contained in an InDataPreprocessList object. The InDataType Class utilizes the file list and data type argument to access data/file type specific information from the InDataTypeTemplate and InFileTypeTemplate classes. The acquired information is used to properly instantiate the appropriate data type subclasses (i.e., InPVMetadata). The instantiated data type subclasses then interact with other classes (i.e., InSourceMCF, DsCIDescriptor, InMetadataTool) to produce final products before insertion into the Data Server Subsystem. The InSourceMCF class provides information on a data type basis on how to retrieve applicable parameter values (e.g., the location of parameter, computer data type). The DsCIDescriptor is a data server class which provides services to acquire a target metadata configuration file and validate metadata. The InMetadataTool Class provides services to read target metadata configuration files and write PVL metadata files. After completion of the preprocessing task, the InDataServer InsertionTask Class interacts with the Data Server Subystem

(DsCIESDTReferenceCollector, DsCIRequest, DsCICommand) to insert the preprocessed data files. Detailed scenarios illustrating interactions between all classes are provided later in this section. With the basic concept of operations in place, the design principles used to develop the object models and individual classes are presented in the following paragraphs.

The Ingest Subsystem will receive data from a variety of external sources and instruments as illustrated previously in Figure 3.1-1. This variety of instruments and sources produce data types which have different file formats (e.g., HDF, SFDU, GRIB), data formats, metadata parameters, and data organizations (e.g., separate file for metadata vs. embedded metadata). This diverse set of ingest data types necessitates different implementations of preprocessing services based on each individual data type or set of data types. For instance, the implementation of conversion services requires different algorithms for data types which possess different initial and/or target formats. In addition, each ingest data type may make use different preprocessing services. For instance, some data types require metadata extraction (e.g., CERES L0 Data, NMC GRIB data), while other data types may provide a separate file which contains only metadata (e.g., TOMS Gridded Ozone, AVHRR Monthly GVI). Therefore, the first major design goal for the Ingest Preprocessing CSC is to provide the capability to preprocess this diverse set of data types efficiently and adequately. The tailoring of preprocessing services to address this diverse set of data types is done by specializing the data type base classes, InMetadata and InScienceData.

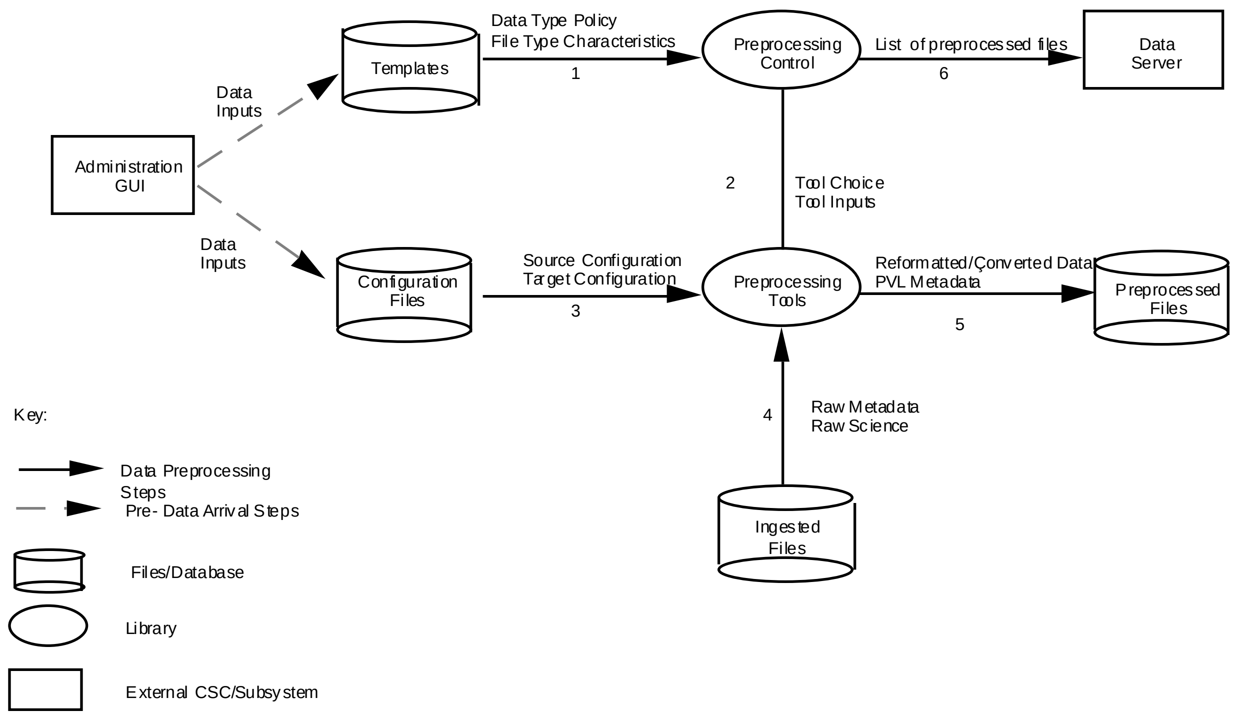


Figure 4.5-5. Ingest Preprocessing CSC Data Flow

The level of specialization for these data type base classes is limited by the second major design goal, eliminating redundancy. The data type subclasses are designed to address groups of data types (where possible) versus a subclass for each data type. If the data types within each group are similar, the services within these subclasses can be tailored to properly preprocess the individual data types through the arguments of the called services. The values of these arguments, for a specific data type, are obtained through the InFileTemplate and InDataTypeTemplate Classes. These "Template" Classes contain information on how to preprocess each specific data type/file type. Part of the information contained in these classes also defines the argument values (dependent on data type) for subsequent use of data type subclass services.

For example, the InPVMetadataClass is a specialization of the InMetadata Class which contains the appropriate preprocessing service for parameter-value metadata. For different data types within this group (parameter-value metadata), different delimiters may be used to separate a parameter from a value. The delimiter for each specific data/file type is defined in the InFileTemplateClass. This information is passed to the InPVMetadataClass upon instantiation. The InPVMetadataClass services will utilize the input arguments to define the appropriate delimiter for a specific service. Whether the delimiter is a semicolon or equal sign does not effect the design of the service. However, it does effect the execution of the service and the resultant product. The delimiter may differ with each data type and therefore must be defined in the input arguments of the service.

This design also helps fulfill the third major design goal, flexibility and extensibility. The CSC is flexible, since modification of existing data types will be feasible. For example, changes in input metadata structure or required metadata output structure are driven by the metadata configuration files. The Data Server subsystem will provide services to access "target" metadata configuration files. These files will dictate the metadata requirements for each data type. Through an iterative process, the preprocessing CSC will retrieve the input values from the input metadata file(s). The InSourceMCF class, provides correlation between the target parameter names in the target metadata configuration file with access information on the input metadata file. The InDataType, InMetadata, InScienceData, and InMetadataTool classes utilize these metadata configuration files to produce final products.

The Ingest Preprocessing classes defining data type (InDataTypeTemplate), file type (InFileTypeTemplate), and metadata configuration (InSourcMCF) will interact with an administration GUI (InTemplateEditor, described in Section 4.5.6, Operator Interface CSC). This will enable operator initiated instantiations of new InFileTypeTemplate, InDataTypeTemplate, InSourceMCF objects to address new data types. The InFileTypeTemplate and InDataTypeTemplate classes within the Ingest Data Preprocessing CSC, will guide the InDataType Class through preprocessing and supplying the necessary information to properly and correctly instantiate the data type subclasses.

The system is extensible, since new data types may be added without major design change. Not only does the basic design often not have to change to accommodate new data types, but some of the preprocessing subclasses may be used for future data types. The implementation of broad based specializations, provides a greater probability that a new data type will fit into an existing data group versus creating a new class for that data type. In addition, the design of these subclasses are a result of Release A and B data sets analysis. Therefore, a Release B data type may fall into one the Release A data groups and require only minimal additional coding. The input arguments would be different (requiring new instances of the "Template" classes; these new instances are created via the InTemplateEditor class) and drive the specialization to properly process the data type correctly. For instance, a Release B data set with Parameter Value metadata can use the services of the existing InPVMetadata Class. A new InFileTypeMetadata object may be created to represent this new data type. It is important to note that not all Release B data types will fit into one of the existing specializations and new data type subclasses will have to be developed in some cases.

The following paragraphs will discuss general preprocessing requirements and how each requirement applies to some of the ingest data types listed in the tables. Table 4.5-3 summarizes ECS data type/set preprocessing requirements for all Release A data types. Items marked as TBR are to be resolved post-Release A CDR with the data providers and processing teams. Release B data type preprocessing requirements will be presented in Release B IDR documentation.

Table 4.5-3. Release-A Ingest Data Type Preprocessing Requirements (1 of 2)

Data Set(s)	File Format	Convert	Reformat	Field/Range Checking	Metadata Extraction
CERES Level 0 Data	SFDU	No	No	Yes	Yes
LIS Level 0 Data	SFDU	No	No	Yes	Yes
TRMM H/K Data	SFDU	No	No	Yes	Yes
TRMM Orbit Data	SFDU	TBR	No	Yes	Yes
LIS SCF Constants and Coefficients	Binary/ASCII	No	No	TBR	TBR
CERES SCF Constants and Coefficients	Native/HDF	No	No	TBR	TBR
CERES SCF Generated Ancillary Data Files	Native/HDF	No	No	TBR	TBR
TSDIS Level 1a Data	SFDU	No	No	Yes	TBR
TSDIS Level 1b-3b Data	HDF	No	No	Yes	TBR
NMC-MRF	GRIB	Yes - to HDF	Yes	Yes	Yes
NMC-ETA	GRIB	Yes - to HDF	Yes	Yes	Yes
NMC-FNL	GRIB	Yes - to HDF	Yes	Yes	Yes
GPCP Data	Native	No	No	Yes	TBR
GPI Data	Native	No	No	Yes	TBR

Table 4.5-3. Release-A Ingest Data Type Preprocessing Requirements (2 of 2)

Data Set(s)	File Format	Convert	Reformat	Field/Range Checking	Metadata Extraction
AVHRR Global Analyzed Field	Binary Direct Access I/O	No	No	Yes	Yes
Snow/Ice Cover (EDR)	Intermediate Database	Yes	Yes	Yes	Yes
AVHRR Monthly General Vegetation Index	Binary Raster Data	No	No	Yes	No
AVHRR Level 1b	TBR	No	No	TBR	TBR
Digital Elevation Map	TBR	No	No	TBR	TBR
Surface Map of Water Conditions	TBR	No	No	TBR	TBR
Surface Map of Vegetation	TBR	No	No	TBR	TBR
TOMS Gridded Ozone	HDF	No	No	Yes	No
SAGE-II Stratospheric Optical Depth	HDF	No	No	No	No metadata exists
SAGE-II Stratospheric Ozone	HDF	No	No	No	No metadata exists
ISSCP DX,D1,D2 radiances	TBR	No	No	TBR	TBR
SSM/I Level 1b	TBR	No	No	TBR	TBR
POAM-II Thin Stratospheric Optical Depth	TBR	No	No	TBR	TBR

Metadata Extraction: The necessity for metadata extraction depends on each individual data type. Landsat 7 Level 0R data does not require metadata extraction since a separate Landsat 7 metadata file is provided. NMC GRIB data requires metadata extraction since the metadata and science data are provided within one physical data file. SDPF Level 0 data provides some metadata in a separate file (Detached SFDU Header) while other metadata (Data Set File Header(s)) are embedded within the science data set file. TSDIS HDF metadata is contained in an separate HDF object from the TSDIS science data. HDF tools will extract the metadata objects from the TSDIS HDF files. TSDIS SFDU metadata is orgainized in a similar format to the SDPF metadata. The necessity for extraction and information, related to where the metadata resides within a physical file, can be found in the appropriate InSourceMCF and InFileTypeTemplate objects.

Conversion: The implementation of conversion services will depend on the initial format of the data type and the targeted archive format. Currently, the three main ingest formats which have been identified for ECS Release A ingest are the: Standard Format Data Unit (SFDU), Hierarchical Data Format (HDF), and Gridded Binary (GRIB). Since HDF is a standard archive format, data conversion from HDF to other formats during the ingest process is not foreseen. SDPF SFDUs (CERES and LIS) will not be converted since the instrument teams will prepare algorithms to process SFDUs as is. TSDIS Level 1A will remain in SFDU format since the TSDIS and ESDIS Projects have jointly determined that it is not efficient or necessary to convert this data to HDF. The lack of efficiency stems from processing complexities with converting TSDIS Level 1A SFDU data into HDF. The lack of necessity is due to TSDIS Level 1A data not being widely distributed to the User community, and therefore a common format such as HDF for this data is not imperative.

An analysis of external ancillary data sets required by the instrument teams has been performed. The result of this analysis was a project directive mandating that only datasets and/or formats required by multiple teams are to be considered for the development of ancillary data reformatting and conversion capabilities. NMC GRIB data, SSM/I snow and ice products, and TOMS products meet this criteria and therefore will be reformatted and converted as needed. NMC GRIB data will require data conversion to HDF-EOS. The detailed design for the NMC GRIB conversion is in progress and will be reported upon post-Release A CDR. Depending on which TOMS product is requried by the CERES instrument there may be TOMS data preprocessing. TOMS Gridded Data is already exists in HDF, while TOMS High Density data will require conversion to HDF-EOS. SSM/I snow and ice products are currently being analyzed. Other CERES ancillary data, GPI, GPCP data will not be converted.

Reformatting: The Ingest Preprocessing CSC will perform reformatting to alleviate inconsistencies in data representations from different external computer systems (e.g., data type definitions, byte ordering, and character representation). HDF data does not have to be reformatted since HDF is a portable file format. An HDF file created on one computer system can be easily read on another computer system without modification. SFDU data will not be reformatted since the data is defined in a byte by byte format. GRIB, SSM/I Snow and Ice Cover, and TOMS data will be reformatted as part of the conversion process.

Metadata Checking: Metadata checking will be performed to verify that certain metadata fields exist in the correct form and that the value of certain fields are within limits. The extent of metadata checking will depend on each data type. The Preprocessing CSC performs some or all of the following field and range checks for metadata on a ingest data type by data type basis:

a. Field Checking. Verifies all required metadata parameters exist and that the metadata parameters use correct syntax.

   ECS has defined a set of "core metadata" which will be supplied for ECS products. This core set of metadata parameters define the minimum metadata which can be supplied for an ECS product. The core metadata will also dictate a specific format and syntax for each parameter. Metadata for a specific data type will have the minimum or core set of metadata and optional metadata data type unique fields. Some data types (e.g., external ancillary data, Level 0 data, Version 0 data) may not be required to meet these core metadata requirements. Level 0 data may not be required to contain this core metadata since it will not be widely accessed by the user community. External ancillary data and V0 migration data may have existing metadata standards and heritage metadata. This does not preclude other subsystems within ECS from generating core metadata for the external ancillary data sets and V0 migration data sets.

The Preprocessing CSC will determine what the required metadata parameters for a specific data type through interaction with the Data Server Subsystem. The Data Server Subsystem supplies a target metadata configuration file dictating the required parameters for the specified data type. A scenario illustrating the full metadata field checking process is included in Table 4.4-10.

b. Range Checking. Verifies for selected metadata parameters, that parameter values lie within a specified range or that the parameters are within a set of discrete values (valid options). The purpose of range checking is not to flag values that may have relevant scientific values outside the normal range of values but to flag fields with values outside of possible range limits or not in a set of possible discrete options. Default values for some selected metadata fields may be inserted when a value is missing.

The Preprocessing CSC will rely on the Data Server Subystem for validation of metadata files. The Preprocessing CSC will first convert the input metadata into a final PVL metadata file and then utilize data server validation services.

c. Byte Checking: In some cases the metadata provided with a specific science data type will indicate the size of the science data packet or file. In this case, the InDataPreprocessTask Class will check that the size of the packet/file is within a certain tolerance.

d. Time Range Checks: Time information in metadata will be checked and translated to the ECS standard format as necessary.

Ingest Unique Metadata: All data types will have ingest unique metadata appended to their original metadata. This ingest unique metadata consists of time of ingest and a quality flag.

4.5.6 Operator Ingest Interface CSC

The Operator Interface CSC provides operations personnel with the capability, via the GUI Interface, to perform physical media ingest; to monitor the ingest history log; to monitor the status of ongoing ingest requests; to update (i.e., cancel, suspend, resume, and change priority) an ingest request; and to set the ingest thresholds. The thresholds consist of: number of ingest requests to be processed concurrently, maximum data volume to be ingested concurrently, Ingest polling timer, and number of data transfer retries. The CSC is comprised of the GUI Interface and the InGUISession, InMediaIngest, InStatusMonitor, InLogMonitor, InRequestController, InThreshold, and InThresholdController object classes. The CSC is initiated as a set of GUI screens with associated software to process GUI input when authorized operations personnel select any of the defined operator service options from the GUI screens. This CSC corresponds to the "Ingest Subsystem Administration Application" CSC described in the SDPS System Design Specification. The Operator GUI Interface contains different screens associated with each operator task. From the GUI screen, the operations personnel select the desired task to be processed and enter the information needed for building the request for the task. The GUI Interface makes sure that all the required fields are properly filled by the operations personnel. If all required fields are indeed properly filled, the GUI Interface invokes the appropriate services provided by this CSC to fulfill the task.

The InMediaIngest, InLogMonitor, InStatusMonitor, and InRequestController object classes are derived from the InGUISession objects class. This means that all of these object classes inherit the data and service members from the InGUISession object class.

The InMediaIngest object class provides services to perform physical media ingest. The object class performs privilege check on both the operations personnel and the media provider. The object class interface with the operations personnel via the GUI Interface.

The InLogMonitor object class provides services to monitor Ingest History Log (an alias for Data Receipt Log). The service allows the operations personnel to specify the search criteria for the Ingest History Log entries for viewing based on 1) ingest start/stop date and time, 2) data provider ID, 3) data set name, and 4) final request status. The InLogMonitor object class is responsible for gathering the search criteria information input by the operations personnel via the GUI Interface and interfaces with the InHistoryLog object class to display the Ingest History Log. The InHistoryLog object class is defined in the Ingest Administration Data CSC; refer to that CSC for details.

The InStatusMonitor object class provides service to monitor the ongoing ingest requests. Authorized operations personnel are allowed to view all or selective ingest requests in the system, whereas a less privileged user can view only the requests that are owned by the individual. The service allows the operations personnel to select Ingest Requests for monitoring based on: 1) data provider ID, 2) ingest request ID, and 3) request state. The object class is responsible for gathering the search criteria information provided by the operations personnel via the GUI Interface and interfaces with the InRequestList to locate the InRequest object class for getting the results. Refer to the Ingest Request Processing CSC for details on the two object classes.

The InRequestController object class provides service to update an ongoing ingest request. The updates include change priority, cancel, suspend, and resume. Only the cancel service is supported at Release A. The object class is responsible for reading in from the operations personnel the update specification (e.g. change priority, cancel) and the request to which the update is to be performed on. It also interfaces with the InRequestList object class to locate the request on which the update is to be performed. Refer to Ingest Request Processing CSC for details on the two object classes.

The InThreshold object class provides services for viewing and updating the values of ingest thresholds. These thresholds consist of the number of ingest requests allowed to be processed concurrently, the volume of data to be ingested concurrently, and the number of data transfer retries.

The InThresholdController object class provides services to set and to view ingest thresholds. The ingest thresholds consist of the number of ingest request to be processed concurrently, maximum data volume to be ingested concurrently, and the number of data transfer retry attempts.

Figures 4.5-6, 4.5-7, and 4.5-8 provide a pictorial overiew of the Media Ingest, the Administrative Viewing, and the Request Cancellation processing respectively. The Administrative Viewing diagram (Figure 4.5-7) describes History Log Monitoring and Request Status Monitoring GUI interfaces in a very generic fashion because of the similarity in their processing--both interface accesses the Ingest DBMS to generate a report for display based on the entries queried from the database.

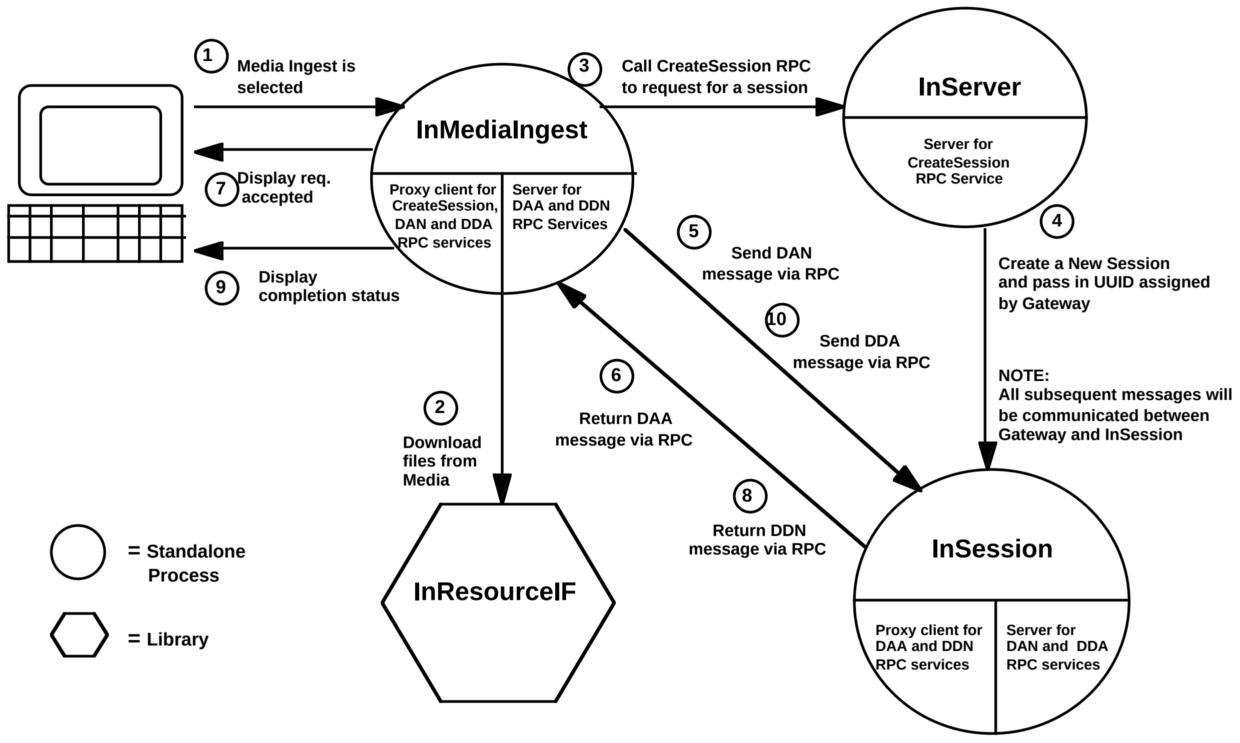

1
Media Ingest is selected
7
Display req. accepted
9
Display completion status
InMediaIngest
Proxy client for CreateSession, DAN and DDA RPC services
Server for DAA and DDN RPC Services
3
Call CreateSession RPC to request for a session
InServer
Server for CreateSession RPC Service
4
Create a New Session and pass in UUID assigned by Gateway
NOTE: All subsequent messages will be communicated between Gateway and InSession
5
Send DAN message via RPC
10
Send DDA message via RPC
6
Return DAA message via RPC
8
Return DDN message via RPC
2
Download files from Media
InResourceIF
InSession
Proxy client for DAA and DDN RPC services
Server for DAN and DDA RPC services
= Standalone Process
= Library

Figure 4.5-6. Media Ingest CSC

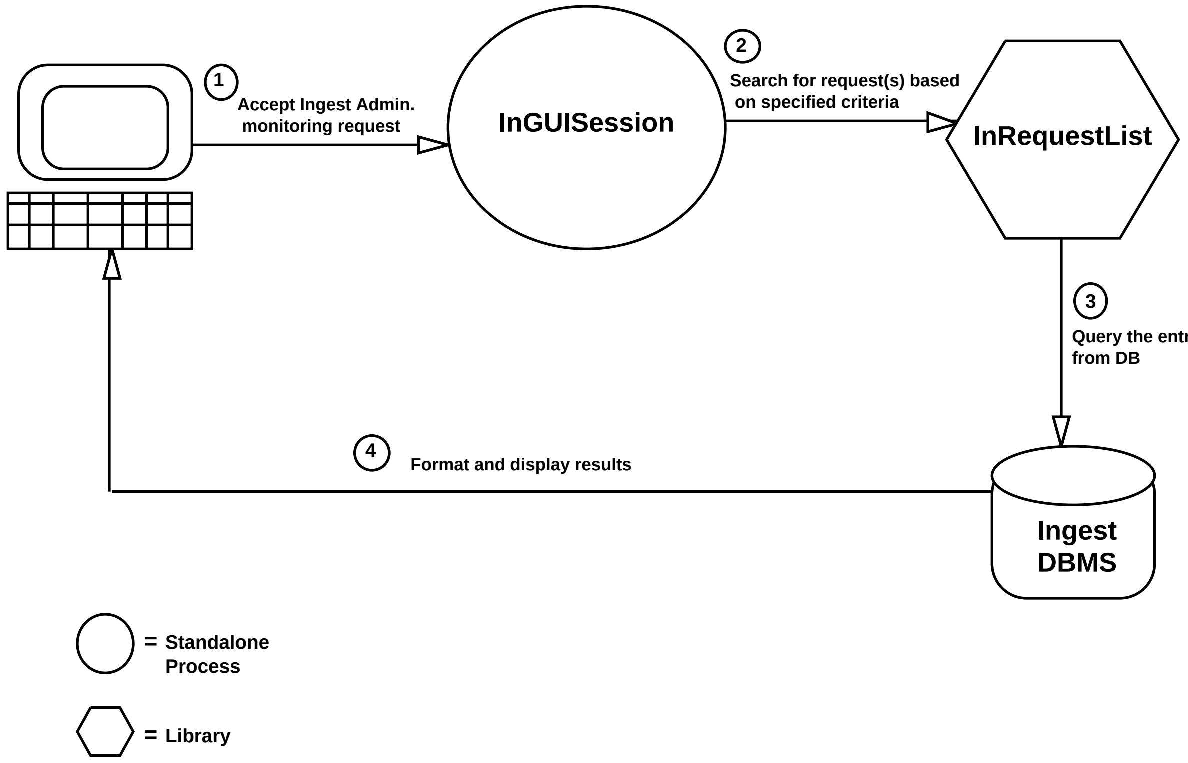

1
Accept Ingest Admin. monitoring request
InGUISession
2
Search for request(s) based on specified criteria
InRequestList
3
Query the entry from DB
4
Format and display results
Ingest DBMS
= Standalone Process
= Library

Figure 4.5-7. Administrative Viewing CSC

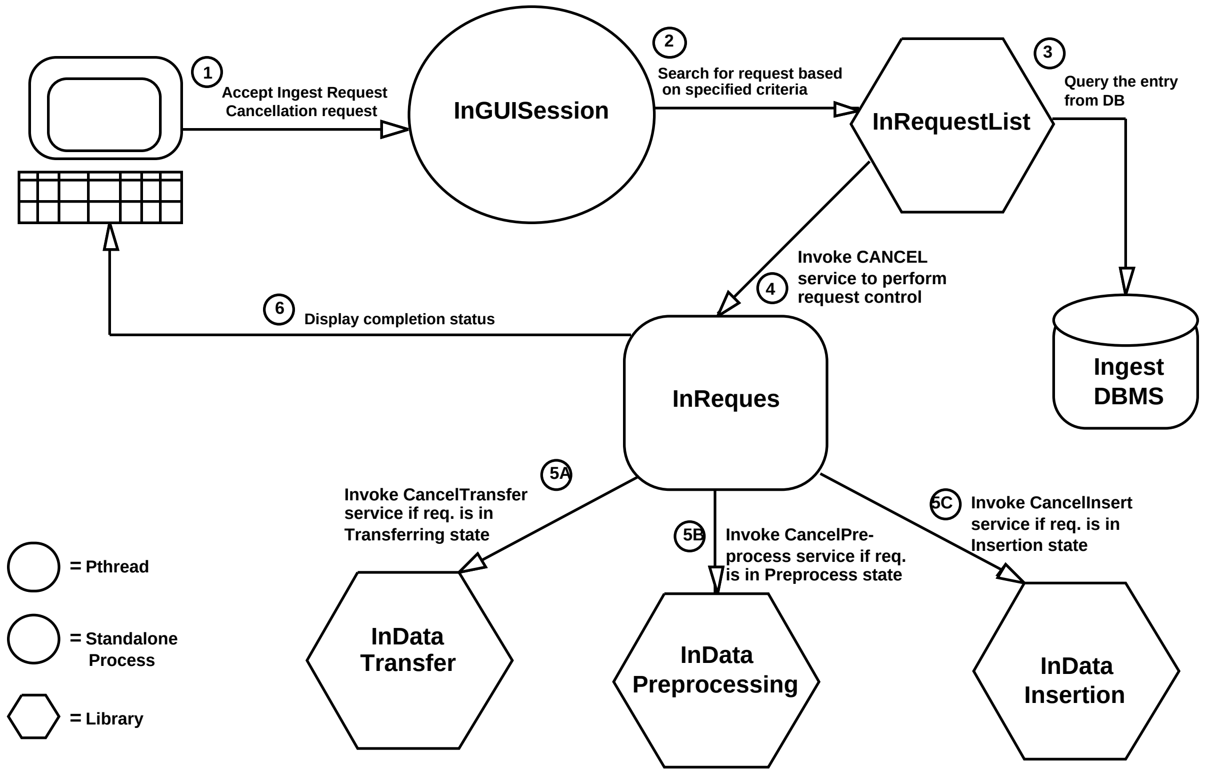

1
Accept Ingest Request Cancellation request
InGUISession
2
Search for request based on specified criteria
InRequestList
3
Query the entry from DB
Ingest DBMS
Invoke CANCEL service to perform request control
4
6
Display completion status
InReques
5A
Invoke CancelTransfer service if req. is in Transferring state
5B
Invoke CancelPre-process service if req. is in Preprocess state
5C
Invoke CancelInsert service if req. is in Insertion state
InData Transfer
InData Preprocessing
InData Insertion
= Pthread
= Standalone Process
= Library

Figure 4.5-8. Request Cancellation CSC

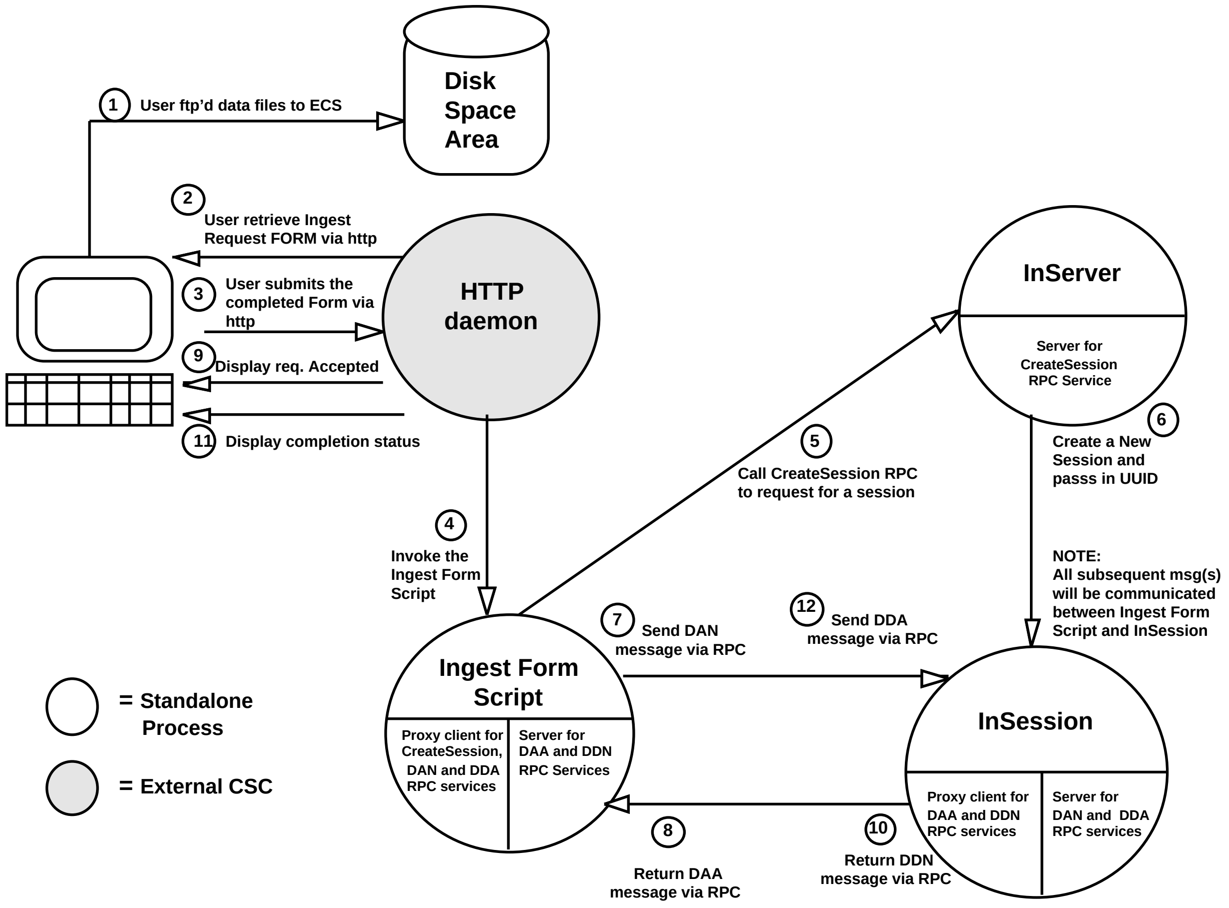
4.5.7 User Network Ingest Interface CSC

The User Network Ingest Interface CSC provides ECS science users with the capability to interactively request network ingest data into the ECS system and to get the status of an ongoing ingest request previously sent by the user. The CSC is composed of GUI Interface; and the InGUISession, InNetworkIngest, and InStatusMonitor object classes. The CSC is initiated as a set of GUI screens with associated software to process GUI input when the user selects the User Network Ingest option from the GUI screens. This CSC corresponds to the "Ingest Preparation Toolkit" CSC described in the SDPS System Design Specification.

Figure 4.5-9 provides an pictorial overview of the User Network Ingest processing. The diagram clearly depicts how a user would ingest data files into the ECS system using HyperText Markup Language (HTML). One thing to be aware of is that before invoking the Ingest Form Script defined in InNetworkIngest object, the fields in the Request Form will be parsed by the HTTP daemon. Upon invocation, the Ingest Form Script will package the Request Form fields into a DAN message and sends the message to ECS Ingest for processing.

The GUI Interface consists of screen(s) associated with the science user task which allows users to interactively fill in information needed for building the network ingest request or the status monitor request. The GUI Interface makes sure that all the required fields are properly filled by the user. If all required fields are properly filled, the GUI Interface invokes the appropriate services provided by the InNetworkIngest or the InStatusMonitor object class to fulfill the request.

The InNetworkIngest and InStatusMonitor object classes are derived from the InGUISession object class. This means that these object classes inherit all the data and service members from the InGUISession object class. The InGUISession object class is defined in the Operator Ingest Interface CSC; refer to the Operator Ingest Interface CSC for details. The InNetworkIngest object class provides services for the user to interactively request ingest of product and document data into the ECS system and to save the contents of the interactively entered network ingest request to a file. The object class is setup to read in the Interactive Network Ingest Request from the user via the User GUI Interface. The InStatusMonitor object class provides services for user to view the state of the user ongoing ingest request(s). The user is authorized to view only the requests that are owned/submitted by the user. The InStatusMonitor object class is defined in the Operator Interface CSC; refer to the Operator Interface CSC for details.



Disk Space Area
1 User ftp'd data files to ECS
2 User retrieve Ingest Request FORM via http
3 User submits the completed Form via http
9 Display req. Accepted
11 Display completion status
HTTP daemon
4 Invoke the Ingest Form Script
5 Call CreateSession RPC to request for a session
InServer
Server for CreateSession RPC Service
6 Create a New Session and passs in UUID
NOTE: All subsequent msg(s) will be communicated between Ingest Form Script and InSession
7 Send DAN message via RPC
12 Send DDA message via RPC
Ingest Form Script
Proxy client for CreateSession, DAN and DDA RPC services
Server for DAA and DDN RPC Services
InSession
Proxy client for DAA and DDN RPC services
Server for DAN and DDA RPC services
8 Return DAA message via RPC
10 Return DDN message via RPC
= Standalone Process
= External CSC

Figure 4.5-9. User Network Ingest CSC

4.5.8 Ingest DBMS CSC

The Ingest DBMS CSC is responsible for storing and providing access to Ingest Subsystem internal data. The CSC is composed of an OTS Data Base Management System (DBMS) . In particular, the DBMS stores the Ingest operations data bases -- Ingest History Logs and Ingest configuration and template information. Those data bases are configured in the Ingest Administration Data CSC, which is described in the following paragraph. This CSC corresponds to the "Subsystem Administration DBMS" CSC described in the SDPS System Design Specification.

4.5.9 Ingest Administration Data CSC

The Ingest Administration Data CSC table descriptions of administrative information for the Ingest operations data bases are maintained by the Ingest DBMS. The Ingest operations data bases are described at a high level in section 4.6.1.4 and at a more detailed level in the object model in Section 4.3. The CSC interfaces with MSS to allow generation of production reports. Refer to Section 4.6.3 for a description of proposed production reports and to the MSS volume of this document for a description of associated MSS event logs.

4.5.10 Peripherals CSC

The Peripherals CSC described in the SDPS System Design Specification is entirely reused from the Peripherals CSC software described in the Data Server Subsystem volume of this document. That CSC provides common access of the Distribution and Ingest Peripheral Management HWCI (DIPHW, as described in volume 9 of this document) for ingest and distribution purposes. The Peripherals CSC is configured on the DIPHW CSCI.

4.5.11 Viewing Tools CSC

The Viewing Tools CSC described in the SDPS System Design Specification is entirely reused from the visualization and other client tools described in the Client Subsystem volume of this document. Viewing capabilities are provided only for ingested data (i.e., only for data already converted/reformatted into forms accessible by standard ECS viewing tools). Data that is not converted/reformatted into a form accessible by standard ECS viewing tools are not available for viewing or other analysis within the Ingest Subsystem. The Viewing Tools CSC is configured on the Ingest Client HWCI (ICLHW) administration workstation component.

4.5.12 Data Storage Software CSC

The Data Storage Software CSC described in the SDPS System Design Specification is entirely reused from the CSCs comprising the Science Data Server (SDSRV) CSCI and Storage Resource Management (STMGT) CSCI described in the Data Server Subsystem volume of this document. Those CSCs provide for reliable storage and retrieval of the Level 0 (L0) data in the same fashion as for other data products. No additional Ingest Subsystem software is required to support L0 high-reliability data storage and subsequent access. The Data Storage Software CSC is configured on the Ingest Client HWCI (ICLHW) host component for Level 0 data ingest.

4.5.13 Resource Administration CSC

The Resource Administration Application CSC described in the SDPS System Design Specification is entirely reused from the administration CSCs comprising the Science Data Server (SDSRV) CSCI and Storage Resource Management (STMGT) CSCI described in the Data Server Subsystem volume of this document. Those CSCs provide for operations staff monitoring and control of storage components. No additional Ingest Subsystem software is required to support L0 high-reliability data storage and subsequent access. The Resource Administration CSC is configured on the Ingest Client HWCI (ICLHW) administration workstation component for Level 0 data ingest.

4.5.14 Client Interfaces CSC

The client CSC is entirely reused from the Science Data Server (SDSRV) CSCI described in the Data Server subsystem volume of this document. That CSC provides public objects/services to perform data granule insertion. The Client CSC is configured on the Ingest Client HWCI (ICLHW) host component for Level 0 data ingest.

4.6 Ingest CSCI Management and Operation

The materials in the following paragraphs discuss the management and operations of software components discussed in section 4.5.

4.6.1 System Management Strategy

The Ingest CSCI is designed to provide robust ingest services to external data providers. Specifically, the design goal of the Ingest CSCI is to always return status (successful or unsuccessful) for every received ingest request. To accomplish that goal, the Ingest CSCI follows ECS project guidelines for:

- Process startup and shutdown;
- Error detection and reporting;
- Fault tolerance and error recovery;
- Ingest operations data bases.

4.6.1.1 Ingest Startup/Shutdown

As described in the MSS volume of this document, MSS provides life-cycle services for system startup and shutdown. The Ingest subsystem fully uses those services.

At system startup, the Ingest InServer object and the InRequestManager object are instantiated as standalone processes. In addition, data base tables corresponding to the InRequestProcessHeader, InRequestSummaryHeader, InRequestProcessData, InRequestSummaryData, and InRequestFileInfo are set up prior to the initial system startup.

InServer acts as an "object factory". InServer instantiates InSession objects as standalone processes when a connection is requested by an External Data Provider. One process is created for each External Data Provider.

InRequestManager also acts as an "object factory." InRequestManager instantiates InRequest objects in process threads (pthreads) when a request is created by an InSession object. The InSession process establishes a client proxy object (InRequest_C) for communicating with the distributed InRequestManager_C (client) object. The InSession object invokes InRequest_C (client) services to create an InRequest_S (server) object. InRequestManager_C passes a creation message to the InRequestManager_S (server) object. The InRequestManager_S object physically creates the InRequest_S (server) object. One InRequest_S object is created for each InSession object.

The InRequestManager process instantiates the InRequest objects in separate pthreads. A pointer to each InRequest_S object is maintained in the InRequestList object, which itself is pointed to by the InRequestManager_S object. The InDataTransferTask, InDataPreprocessingTask, and InDataInsertTask objects are all instantiated within the context of the InRequest pthread.

When Ingest processes are started, they check for the existence of checkpointed information. If such checkpointed information is available, the Ingest processes will restore the information and continue processing. Note: there is a tunable time limit after which checkpointed information will not be restored. Additional checkpointing strategy is discussed in section 4.6.1.3.

4.6.1.2 Error Detection and Reporting

As described in the CSS and MSS volumes of this document, CSS and MSS jointly provide event logging services for logging and reporting errors and faults, and for browsing error/status logs. The Ingest subsystem fully uses those services. Errors detected in the processing of the InRequest object are identified in Table 4.6-1, which shows critical errors reported and the actions (including operations personnel actions) taken.

Table 4.6-1. Ingest Subsystem Error Categories (1 of 3)

Error Category	Actions to Be Taken
Metadata Validation Failure	In general, log errors to the event log and return status to the external data provider. Operations staff evaluate errors off-line and request re-ingest after corrections. Dependent on preprocessing templates (instituting DAAC policy), continue with preprocessing and insertion of data into the data server subystem in a special data type category, "INCOMPLETE" (with metadata quality set to "UNVALIDATED"). Flag Metadata indicating that the metadata was not validated. Operator action and source facility notification required. Log errors to the event log and return status to the external data provider. Operations staff evaluate errors off-line and request re-ingest as necessary.
Target/Source MCF Mismatch Failure	Same as above. Operations staff are alerted that preprocessing templates are out-of-synch. Operations staff develop a trouble ticket to correct the mismatch.
Data Type Policy Failure:No Metadata	Log errors to the event log and return status to the external data provider. Operations staff evaluate errors off-line and request re-ingest after corrections.

Table 4.6-1. Ingest Subsystem Error Categories (2 of 3)

Error Category	Actions to Be Taken
Data Type Policy Failure: Incomplete set of Files, Meta-data exists	In general, log errors to the event log and return status to the external data provider. Operations staff evaluate errors off-line and request re-ingest after corrections. Dependent on preprocessing templates (instituting DAAC policy), continue with pre-processing and insertion of data into the data server subystem in a special data type category, "INCOMPLETE" (with metadata quality set to "UNVALIDATED"). Flag Metadata indicating that the metadata was not validated. Operator action and source facility notification required. Log errors to the event log and return status to the ex-ternal data provider. Operations staff evaluate errors off-line and request re-ingest as necessary.
Data Type Policy Failure: Unknown Data Type	Log errors to the event log and return status to the external data provider. Opera-tions staff evaluate errors off-line and communicate with external data provider. (Note: all data sets to be ingested must be pre-authorized by DAAC personnel. In-gest and Data Server setup is required before ingest and archive can occur.)
Data Type Policy Failure: Unknown File Type.	Same as above.
File Type Failure	In general, log errors to the event log and return status to the external data provider. Operations staff evaluate errors off-line and request re-ingest after corrections. Dependent on preprocessing templates (instituting DAAC policy), continue with pre-processing and insertion of data into the data server subystem in a special data type category, "INCOMPLETE" (with metadata quality set to "UNVALIDATED"). Flag Metadata indicating that the metadata was not validated. Operator action and source facility notification required. Log errors to the event log and return status to the ex-ternal data provider. Operations staff evaluate errors off-line and request re-ingest as necessary. Operations staff are alerted that preprocessing templates are out-of-synch. Opera-tions staff develop a trouble ticket to correct the mismatch.
Unable to archive data	Internal Data Server fault. Log errors to the event log and return status to the exter-nal data provider. Report alert to operations staff. Operations staff evaluate errors off-line and request re-ingest as necessary. (Note: Data Server will re-vector ingest-ed data to a different device in the event of a single device failure.)
Unable to read pe-ripheral media	Internal Data Server fault. Log errors to the event log and return status to the exter-nal data provider. Report alert to operations staff. Operations staff evaluate errors off-line and request re-ingest as necessary. (Note: if a peripheral device fails, Data Server will revector the media to a different peripheral.)
Unable to transmit data ingest coordi-nation messages	After a system-tunable number of retries, log errors to the event log. Report alert to operations staff. Operations staff evaluate errors off-line to evaluate and correct communications network problems.
Unable to transfer data to be ingested	After a system-tunable number of retries, log errors to the event log and return status to the external data provider. Report alert to operations staff. Operations staff eval-uate errors off-line to evaluate and correct communications network problems.

Table 4.6-1. Ingest Subsystem Error Categories (3 of 3)

Error Category	Actions to Be Taken
Unable to allocate disk space	Unable to allocate working storage space using Data Server STMGT CSCI services. Log errors to the event log and return status to the external data provider. Report alert to operations staff. Operations staff evaluate errors off-line and request re-ingest as necessary.
Unable to set up external data provider session	Limit exceeded for allowable number of external data provider sessions. Log errors to the event log and return status to the external data provider, indicating that the session connection should be re-attempted later. (Note: based on the modeled transaction load, this error condition is expected to occur very rarely, if at all.)

Errors/status may be reported in two error logs. MSS maintains the first log, the MSS event log. It contains errors/status of interest to operations staff to evaluate system status and to perform trend analysis. The Ingest subsystem maintains the second log, the Ingest event log. The Ingest event log contains selected errors/status from the MSS event log (for context) plus highly-detailed debug events. Software maintenance personnel use the Ingest event log to diagnose system and software problems in response to trouble tickets.

4.6.1.3 Fault Tolerance and Error Recovery

Once an ingest request is accepted from an External Data Provider, the ECS policy is to complete request processing and return status (successful or unsuccessful) to the External Data Provider. Therefore, upon creation of the InServer, InSession, InRequestManager, and InRequest objects, their critical attributes are checkpointed to a COTS data base. After a process or system failure, the checkpointed attributes are automatically restored to the last checkpointed state and processing continues. Note: there is a tunable time limit after which checkpointed information will not be restored.

Data bases for InServer/InSession, and InRequestManager/InRequest are checkpointed only when a session or request, respectively, changes its fundamenal state. Any change to state that does not invoke checkpointing is defined as a "substate."

The InServer object has two fundamental states--"Session Created" and "Session Deleted." The InServer object is checkpointed after each InSession object is created (storing session ID) and after each InSession object is deleted (deleting session ID). The InServer object has two substates--"Active" (at object creation) and "Inactive" (at object deletion) that are not checkpointed.

The InSession object has three fundamental states--"Active", "Request Created", and "Request Deleted." The InSession object is checkpointed immediately after it is created (storing information about the connection with the external data provider); after a InRequest object is created (storing the request ID); and immediately after the InRequest object is deleted (deleting request ID).

The InRequestManager object has two fundamental states--"Request Created" and "Request Deleted." The InRequestManager object is checkpointed after each InRequest object is created (storing request ID in the InRequestList object) and after each InRequest object is deleted (deleting request ID from the list). The InRequestManager object has two substates--"Active" (at object creation) and "Inactive" (at object deletion)--that are not checkpointed.

The InRequest object has three fundamental states--"Active", "Data Submitted for Insertion", and "Data Inserted". The InRequest object is checkpointed to "Active" when created by InRequestManager. The InRequest object is subsequently checkpointed when a data granule is submitted to the Data Server for insertion and when the insertion is complete. The InRequest object has two substates--"Data Transferred" and "Data Preprocessed"--that are not checkpointed.

Note: therefore, after process or system failures between InRequest creation and submittal of data granules for insertion in the Data Server, the InRequest object is restored as if no data transfer or preprocessing had occurred. Temporary files are deleted based on checkpointed file IDs.

Failure scenarios with recovery methods:

1) Failure of the InServer object. This process is immediately restarted by MSS restart services as a Unix standalone process. The list of active session IDs is restored from its checkpointed state. Corresponding InSession processes are restarted if they were disabled. Prior to restart, no External Data Providers may set up new connections (connection requests are rejected). Prior to restart, InSession objects that have completed all interactions with an External Data Provider are blocked from terminating.

2) Failure of an InSession object. The InServer process detects the failure via a system signal. InServer creates a new InSession object that communicates with the External Data Provider and with the submitted InRequest based on the checkpointed list of request IDs (OODCE OID). Prior to restart, the External Data Provider application may time out. The InRequest object that has subsequently completed is blocked until the InSession object is restored. No other ingest sessions that have not failed are affected.

3) Loss of the data base tables used for checkpointing. The data base management system (DBMS) automatically checkpoints transactions to allow restorations of table information. In the event of a hard disk failure/DBMS failure, the InRequest object detects the failure, and reports a request failure condition to the InSession object.

4) Failure of InRequestManager. All associated InRequest pthreads are terminated. The InRequestManager process is immediately restarted as a Unix standalone process. The identifiers of InRequest objects are restored to the InRequestList. The InRequestManager recreates the InRequest objects and they continue processing from their latest checkpoint (see the next scenario).

5) Failure of individual InRequest pthreads. The InRequestManager process detects the failure and creates a new InRequest object that communicates with the originating InSession object based on the checkpointed request ID (OODCE OID). The InRequest object reverts to its latest checkpoint, cleans up temporary files, and continues processing from the checkpoint. No other ingest requests that have not failed are affected. Note: as described earlier in paragraph 4.6.1.3, data is re-transferred from the external data provider if data has not been submitted to the Data Server for insertion.

6) Failure of the processor on which an Ingest process is running. In general, the processor automatically restarts. Restart of individual processes is handled as a combination of one or more of the above process restarts.

If the processor is disabled, the disablement is detected by MSS SNMP services and, where available, a backup processor is restarted. The backup processor has full access to the data base tables used for checkpointing and has an identical network address to that of the primary processor.

Again, restart of individual processes is handled as described above. Note: in the Ingest subsystem a backup processor is generally provided only in the case of Level 0 data ingest, where a separate set of Ingest Client processors is provided for high availability.

7) Failure of the External Data Provider application. After a given number of retries to transmit the Data Delivery Notice (DDN), operations staff are notified by means of an alert message. The Data Delivery Acknowledgement (DDA) is simulated for InSession, which exits normally. The DAAC operations staff will coordinate with the External Data Provider operations staff to diagnose the failure.

8) Network failure for the connection with the External Data Provider application. Same as 7).

4.6.1.4 Ingest Operations Data Bases

The Ingest CSCI maintains two types of operations data bases:

- Ingest history logs
- Ingest configuration and template information

Ingest history logs contain detailed and summary information about ingest request status. The two types of information are complementary and fulfill different operational requirements. Both types of information are stored in a COTS data base management system.

The detailed information is used to restore system state after a process or system failure as described above. Selected information (e.g., active request IDs, request state) is available for display of status on ongoing requests. The detailed information is deleted when processing for a request is complete. In the object class descriptions in Section 4.3, classes In Request Process Data, In Request Process Header, and In Request File Info contain the detailed information.

The summary information is used to maintain longer-term summary statistics on ingest processing. Selected information (e.g., request IDs, data volume ingested, number of data granules successfully/unsuccessfully ingested) is available for display. The summary information is updated during ingest request processing and maintained after request processing is complete. Summary information is deleted periodically based on DAAC operational policy. In the object class description in Section 4.3, classes In Request Summary Header and In Request Summary Data contain the summary information.

Ingest configuration and template information is stored by operations personnel to implement data-specific system functions and to fine-tune system performance. Operations personnel are responsible for entering and updating templates that guide ingest data processing. Separate templates are implemented for data type information, file type information, and metadata configuration information. Operations personnel are also responsible for entering and updating tunable parameter information. Parameters for ingest thresholds (e.g., maximum number of requests to concurrently process, maximum data volume to concurrently process, number of data transfer retries) allow operations personnel to throttle ingest processing and the flow of data into site Data Servers. In the object class descriptions in Section 4.3, classes In Data Type Template, In File Type Template, and In Source MCF contain template information. In System threshold, In External Data Provider threshold, ane In Polling threshold contain tunable parameters.

Operations interfaces to the operations data bases are discussed in section 4.6.2. Additional uses of the operations data bases are discussed in Section 4.6.3.

4.6.2 Operator Interfaces

The Ingest GUI interface provides a collection of GUI components through which privileged users (e.g., operator) can access services in ECS Ingest. The Ingest GUI interface is categorized into three groups: Administration, Media Ingest, and User Network Ingest.

The Administration GUI interface provides operations personnel with the capability to monitor and control all of the Ingest activities in the system; and to input or update the data validation templates. The interface is composed of five GUI screens.

The first Administration GUI interface is Request Status Monitoring, which allows operations personnel to selectively choose requests for viewing their current processing state based on the specified criteria. The second Administration GUI interface is Request Control. Operations personnel can use this GUI interface to delete, cancel, suspend or resume requests; suspend and resume are post Release A functions. Next is the Threshold Control GUI interface. It allows operations personnel to view or to update the Ingest Thresholds. The thresholds consist of 1) the maximum allowed Ingest Request to be processed concurrently, 2) the number of transfer retry attempts when network failure occurs, 3) the Polling timer indicating how long to wait before starting the Ingest Polling, and 4) the maximum allowed data volume to be processed concurrently. The next Administration GUI interfaces is Ingest History Log Viewing. The History Log provides a summary of Ingest activities that have happened. The log entries can be queried through a selection of criteria. The last Administration GUI interface is Data Validation Template Update. This interface allows operations personnel to define and to update the data/file type policy and the source/target format information. These templates are needed by the Ingest to perform appropriate data validation and preprocessing.

The Media Ingest GUI interface allows operations personnel to perform physical media Ingest. The type of media supported in physical media ingest in Release A is 8mm tape.

The User Network Ingest GUI interface allows privileged users to electronically ingest data into the ECS system. There are two types of GUI screens for the User Network Ingest; one type for science data ingest and the other type for document ingest. In addition to the ingesting service, both types of user ingest GUI interface provides users the capability to view the status of submitted ingest requests. This service allows users to find out the processing state of a previously-submitted request.

All of the Ingest GUI interfaces will be implemented based on the "ECS User Interface Style Guide" document. The document defines standards for the ECS User Interface design and implementation. The intent of using the guide for GUI development is to help ensure that ECS has a consistent and common look and feel user interface system wide.

The Administrative and the Media Ingest GUI interface will be developed in X-Windows/Motif. For the User Network Ingest service, it will be developed inHyperText Markup Language (HTML).

Table 4.6-2 provides a summary of all the Ingest GUI components.

Table 4.6-2. Ingest GUI Screens (1 of 2)

GUI	Input	Output
Request Status Monitoring	Search criteria (1 only): -A specific Request Id -User Id for all associated Requests -All Requests	Request info: -External Data Provider -Request Id -Total Ingest Data Volume -Request State
Request Control	Request Id Control Type list (1 only): -Cancel -Hold -Resume -Change Priority If Change Priority selected: New Priority level	Completion status
Threshold Control	Threshold Type List (choose 1): -Max Request -Max Request for each External Data Provider -Max Total Data Volume -Max Data Volume for each External Data Provider -Max transfer retries -Polling Timer New value	Completion status
Ingest History Log Viewing	Search criteria (>= 1): -Start Time/Stop Time -External Data Provider -Data Type -Final service status	Event entries from the History Log: -Ingest start/stop date and time -Request Id -External Data Provider -Final Service Status -Data Type -Ingest Data Volume -# granules/datasets -# of files
Data Validation Template Update	Template Type List (choose 1): -Data Type Template -File Type Template -Source MCF Template Update Type (choose 1): -Update Template Entry -Add Template Entry -Delete Template Entry If Update or Add selected: New Template Entry Info	Completion Status

Table 4.6-2. Ingest GUI Screens (2 of 2)

GUI	Input	Output
Media Ingest	Media Type Total Media Count Media Volume Media Data Provider	Request Id Processing State (inform periodically)
User Network Ingest	by User: -Data Type -File List -Data Avail date/time -Desired Priority by System: -External Data Provider/User Id -File Size OPTION: -Save screen request to file	Request Id Processing State (inform periodically)
User Document Ingest	Same as above	

4.6.3 Ingest Production Reports

In addition to ad hoc ongoing request status displays discussed above, the Ingest subsystem provides the standard reports described in Table 4.6-3.

Table 4.6-3. Standard Ingest Production Reports

Report Type	Report Description	Intended Audience
Ingest History Report (summary and detailed versions)	The report gives a detailed account of all ingest requests processed during a reporting period, as well summary statistics. The report supplies operations staff with a log and summary view of ingest request completion performance (e.g., maximum, minimum, average volume of data ingested). The report is generated on a daily, weekly, and monthly summary basis, and as an ad hoc report.	Ingest Technician Resource Planner Performance Analyst Operations Supervisor DAAC Assistant DAAC Manager
Ingest Error Report	The Ingest Error Report is a summary report of the frequency of errors of different types encountered during ingest processing. The report consists of two sections--Data Set Summary and Error Class Summary. The Data Set Summary lists a count of reported errors, by error class, for each data set type. The Error Class Summary lists a count of reported errors for each error class.	Same as above

This page intentionally left blank.

5. ICLHW - Ingest Client HWCI

5.1 Introduction

Overall Ingest Subsystem responsibility incorporates the support of multiple functions within ECS including user data ingest, hard media ingest, and the monitoring of ingest status. The principal area of ICLHW responsibility in the SDPS architecture is the ingest and storage of Level 0 (L0) data sets for a period of one year. The criticality of the timely and reliable storage of L0 data sets demands that a separate high reliability, high availability data server instantiation be dedicated to L0 data ingest. Data which has been ingested will be made available to the Processing Subsystem for the generation of higher level products. The daily volume of L0 data to be ingested at each DAAC is taken from the 6/21/95 ECS Technical Baseline, and is shown in Table 5.1-1. Dates listed at the top of this table correspond to dates where a change to the mission baseline occurs. The Data Server Volume of this document indicates those data products which will be ingested directly into hardware provided by the Data Server Subsystem utilizing ingest client software. Sizing of Ingest Subsystem resources is sufficient to accommodate the listed volumes, as described in the following paragraphs. The daily volumes to be ingested change at each ECS release, generally increasing as additional missions are supported. The Ingest Subsystem design is therefore scaleable to support the growing daily and total data volumes.

Functionality similar to that provided by the ingest client hosts is responsible for the ingest of data types other than L0 in the Data Server Subsystem. Ingest client software capabilities are mapped to data server components for support of the ingest of data types that are stored permanently within the data server. The ingest and archiving of these other data types is discussed in the Data Server Subsystem description volume of this document.

Table 5.1-1. Daily L0 Ingest Volumes

Level 0 GB/day	9/30/97	9/30/98	12/31/99	10/31/01	12/31/02	12/31/03
ASF						
EDC*	0.00	140.00	140.00	140.00	140.00	140.00
GSFC	0.00	66.96	67.57	149.95	150.43	151.51
JPL	0.00	0.00	0.07	0.07	0.07	0.07
LaRC	0.25	41.54	41.55	42.25	77.24	77.24
MSFC	0.26	0.26	0.26	0.98	0.98	0.98
NSIDC						
ORNL						
Sum (GB/day):	0.51	248.76	249.45	333.25	368.72	369.80

* Landsat-7 L0R data at EDC enters the SDPS through the Ingest Subsystem and is archived within the Data Server Subsystem

5.1.1 HWCI Design Drivers

The Ingest Subsystem hardware components consist of the client host servers, working storage, and L0 archive repository. The rationale for the separation of L0 data ingest functions from other data server functionality is discussed in Section 3.2.2. The data rates and volumes to be supported at each of the DAAC sites varies, but the basic Ingest Subsystem configuration is consistent at all sites and is described in the following section. The HWCI for the Ingest Subsystem consists solely of the Ingest Client HWCI (ICLHW), which is comprised of the ingest client hosts required for ingest management, control, monitoring, and processing of ingested data. The ICLHW also contains working storage and archive repository components similar to those of the data server Working Storage HWCI (WKSHW) and Data Repository HWCI (DRPHW). Those components utilized for the ICLHW are specialized for use in the Ingest Subsystem due to the unique ingest reliability, maintainability, and availability requirements as discussed in Section 3.2.2.1.

The client hosts manage the transfer of data into, out of, and within the Ingest Subsystem. The loading on the client hosts is principally in two areas: I/O loading associated with the ingest of Level 0 data from external sources and CPU loading required to perform basic ingest data checks, metadata validation, and metadata extraction. Additional functions to be performed include logging, status, and reporting activities, coordination of data transfers between working storage and the ingest L0 archive, maintaining a database of all data contained within the Ingest Subsystem, and servicing queries and retrievals on the archived L0 data. The principal factors affecting sizing of the client hosts are summarized in Table 5.1-2.

Table 5.1-2. Ingest Client Sizing Factors

Sizing factors	Driving requirements
Ingest client I/O	• Receipt of L0 data from multiple external sources • Servicing L0 archive queries • Support of CPU loading factors
CPU loading	• Data reformatting • Support of ingest I/O loads • Ingest data server database maintenance • Ingest data checking • Metadata validation and extraction

5.1.1.1 Key Trades and Analysis

One trade analysis specific to the Ingest Subsystem hardware configuration was performed prior to the ECS CDR. The ECS Ingest Subsystem Topology Analysis, ECS Technical Paper 440-TP-014-001, analyzes and documents the configuration of the Ingest Subsystem based on the results of the ECS Ingest Subsystem Design Analysis delivered at PDR. The topology analysis focuses on factors that affect the sizing requirements of Ingest Subsystem components, as well as the capability of the configuration to meet reliability and availability requirements. Methods for assuring acceptable system RMA, as well as the flexibility to evolve the ingest configuration as mission requirements change are discussed as key Ingest Subsystem design drivers.

The principal output of the study is the sizing of Ingest Subsystem components at each of the Release A DAACs. The component sizing was accomplished using a combination of a paper analysis of storage required to satisfy daily and annual ingest volumes, plus the development of a queingmodel showing several system loading parameters at each stage of the ingest process. An additional result of this study is that the implementation of the Ingest Subsystem architecture presented at PDR will satisfy the RMA and data ingest requirements based on an analysis of key Ingest Subsystem drivers in Section 3.2. The implementation of this architecture utilizing redundant ingest client hosts in a prime/backup configuration, as well as high reliability RAID storage devices will provide the system availability and scaleability necessary to support the reliable ingest of data. Outputs from this trade study have been incorporated into the design information presented in this document.

5.1.1.2 Ingest HWCI Sizing and Performance Analysis

The sizing of Ingest Subsystem hardware both from a system level and a component level is based on the 6/21/95 version of the ECS Technical Baseline. Among the information included in the baseline is data by instrument, average daily data volume by level, and data destination. The average expected daily and annual data volumes at each site were calculated from this information and used to determine the required ingest hardware capabilities. Ingest client hosts are sized to accommodate the required ingest volumes as well as I/O and CPU capabilities to support internal data transfers associated with metadata validation and extraction, L0 archive data retrieval, and transfer of data to the Data Server or Processing Subsystems. Working storage disks are sized to accommodate the above functions, as well as provide contingency space for the transfer of more than one days worth of data within a 24-hour period. Working storage space is also effectively increased by the use of a separate L0 rolling store, to which ingested data is written as soon as required metadata extraction and validation is performed. The ingest L0 archive is sized to store one years worth of L0 data, with sufficient storage and I/O capabilities to support anticipated archive write and read loads. Since high RMA is a driver for the Ingest Subsystem, all critical components also include some type of sparing or redundancy to ensure that availability requirements are met.

An Ingest Queuing Model (Imodel) was developed to assist in the sizing of Ingest Subsystem components, and is a very high level look at the data flows in the Ingest subsystem. This analysis is valid for systems where each queue has only one server, and the inter-arrival times and service times have exponential probability densities. In addition, the results are only valid for the steady-state conditions, where the probability of finding the system in a given state does not change with time.

Model output is dependent on a series of model input parameters that may be varied depending on characteristics of the data to be ingested, processed, and stored, as well as network and Ingest Subsystem component capabilities and performance. Parameters such as server CPU and I/O performance, disk I/O, network performance, and the number of operations/byte associated with each server process (e.g., capture, format, archive, distribute) may all be varied to analyze the sensitivity of changes in data flows and system architecture. The load presented by each flow in pkts/sec is a function of the number of bits/sec input from the previous process and the mean size of packet/data set that this process expects. The “Source” item sends packets whose size and rate depend on the network technology chosen. FDDI is baselined at the Release A sites, with an assumed network efficiency of 60% from the maximum 100 Mbps clock rate. The "capture" item includes server functions that receive data from the network or hard media device. Capture rate is a function of the input bandwidth and server I/O capabilities. The "work.store IN" includes the writing of the ingested data to working storage and "work.store OUT" models the writing of data back to the server to perform the "format" operation. Both read and write operations have associated transfer rate and access time estimates for each data transfer. Conservative estimates of 2 MB/sec read and write rates are used based on results from Data Server prototyping efforts. Writing of data to the L0 rolling store archive involves the capabilities of the working storage disks, server I/O, and rolling store devices. Finally, the number of copies of data read from the archive and sent to a data sink (e.g., Processing Subsystem) may be varied to determine the additional load of reprocessing on Ingest Subsystem resources.

Competitive procurement restrictions constrain this paper to identification of the class of component and general performance characteristics used in the analysis, rather than actual candidate components. The client host CPUs are low to mid-level Symmetric Multi-Processing (SMP) 32-bit machines, capable of supporting multiple network (FDDI) and direct-connect (SCSI II) devices. Working storage devices are RAID 5 units with a minimum of 2.0 days worth of space allocated to ingest working storage required to support the functions of acquiring, processing, validation, and archiving L0 data. This volume of working storage allows for one days worth of L0 data to be staged for processing, an additional days worth available for subsequent ingest, and an additional 25% available to service additional Ingest Subsystem needs (e.g., L0 archive retrieval support, pre-processing, quality checking). Additional magnetic disk resources are supplied within the Ingest Subsystem to support items such as client host operating system and application software and L0 archive database directory information. The L0 rolling store may be implemented in several ways, subject to additional analysis prior to the delivery of the final version of this technical paper. Additional RAID was originally planned to function as the L0 rolling store, as the annual volumes at LaRC and MSFC were in the 30-35 GB range. Recent inclusion of TRMM spacecraft housekeeping packets in the volume to be stored increases the annual volumes to the 90-95 GB range. A study has been initiated to compare cost, performance, reliability, and other issues associated with adding more RAID versus the use of an alternate archive device (e.g., 3590 drives in a small stacker configuration). The results of this analysis will be presented at CDR if the results indicate that replacement of some of the RAID with another storage media is a technically desirable and cost-effective option.

Analysis of the ingest queuing model developed for CDR confirms the results of the paper analysis conducted prior to PDR supporting the ability of the Release A Ingest Subsystem configurations to support Level 0 data ingest requirements. Projected maximum individual component utilization in support of Level 0 data ingest is less than ten percent under nominal operations. The additional subsystem capacity is available for contingency (e.g., ingest of more than one days worth of data in one day or resolving difficulties encountered during reformatting or metadata validation activities), Version 0 data ingest, and subsystem testing requirements. The DAAC-specific volume presents additional Ingest Subsystem hardware sizing detail and rationale.

5.1.1.3 Scaleability, Evolvability, and Migration to Release B

Ingest Subsystem hardware must be easily scaleable to support both different subsystem capabilities at each of the DAAC sites as well as changing data ingest requirements over the life of the EOS program. The architecture accommodates the required scaleability through several different mechanisms within the Ingest Subsystem components as described below.

Ingest Client Hosts - Client host resources may be increased in two ways as support for the number and complexity of ingest clients increases. The first is through the addition of CPU, internal memory, and I/O capabilities within a given client host platform or family. The second is the addition of additional client host platforms to support additional interfaces and data providers as the ingest client load increases. Client hosts can be added virtually without limit as network attached storage devices are incorporated into the architecture. Release A client hosts that are planned to be selected will possess sufficient processing and I/O capabilities so that it will not be necessary to upgrade them at Release B at some of the DAACs. Machine upgrade or exchange for a larger class machine may be necessary where processing and I/O demands for a single interface increase significantly.

Working Storage - Local RAID disk banks can be added as working storage needs increase. The limiting factor becomes the number of client hosts that can be directly connected to a particular disk bank. Later ECS releases may also use network attached storage to facilitates the addition of disk space: Disks can be added to the network connections, and when the networks begin to become saturated, segmentation and subnetting techniques can improve performance.

L0 Archive Repository - The L0 archive repository is scaleable much like the client hosts and working storage components in that additional media drives within a repository or additional repositories may be added as required storage volumes increase. The ability to use multiple file servers and the mixing of storage solutions like FSMS based systems in the architecture lends itself to easy scaling. Adding additional file servers can enhance both file handling bandwidth and processing capabilities. The flat common access nature of the robotics based tape repositories allows the adding of file servers to gain access to the data without burdening existing file servers.

5.1.2 HWCI Structure

The Ingest HWCI consists of all of the hardware contained within the Ingest Subsystem. Client host computers control the flow of data into, out of, and within the Ingest Subsystem. Working storage magnetic disks provide temporary storage for ingested data. The L0 archive repository contains the devices required at each site for storage of Level 0 data for a period of one year. These are the principal components which comprise the Ingest HWCI, and are described in greater detail in Section 5.1.2.2. Other peripheral devices such as administrative terminals or workstations are required for Ingest Subsystem and overall SDPS control, but are not strictly part of this HWCI. The Ingest Subsystem may also make use of hardware contained within other HWCIs, such as the Distribution and Ingest Peripheral HWCI (DIPHW) components in support of the ingest of data from hard media. Figure 5.1-1 shows the generic Release A Ingest HWCI configuration. DAAC-specific instantiations of this configuration are contained in the DAAC-specific configuration volume.

5.1.2.1 Ingest HWCI Connectivity

The ingest clients support several different classes of interfaces. These interfaces support ingest of data, insertion of the data into the ingest L0 archive repository, and control and monitoring of Ingest Subsystem resources. The interface classes are summarized in Table 5.1-3.

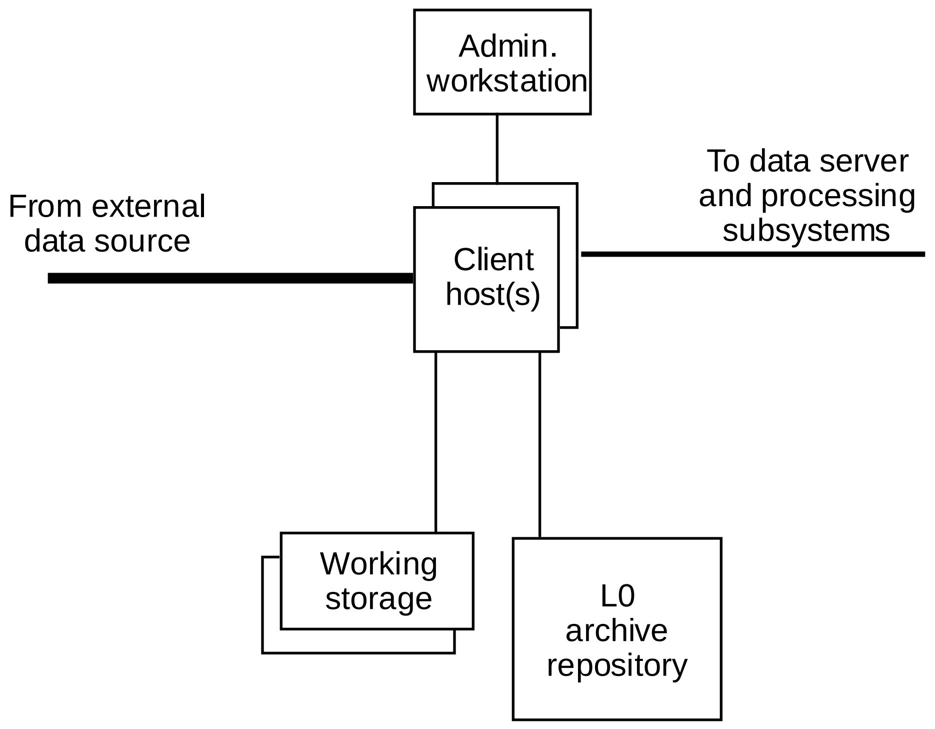


Figure 5.1-1. Ingest HWCI Block Diagram

Table 5.1-3. Ingest Client Interfaces

Ingest Client I/F	I/F Type	I/F Instantiation Examples
External data ingest	Network	FDDI
Ingest peripherals (hard media ingest)	Network/bus	SCSI II
Ingest working storage	Network/bus	SCSI II
Ingest L0 archive repository	Network/bus	SCSI II
Client host control/monitoring	Network	FDDI, Ethernet
Ingest Subsystem control/monitoring	Network	FDDI, Ethernet

Some ingest data sources will provide data via different types of hard media, but this is generally limited to contingency transfers and possibly some Version 0 migration data in Release A. The majority of the Release A data transfers will occur through an electronic interface. The mechanism for electronic transfer also varies depending on the data source. Some data sources will notify ECS of the availability of data, which the Ingest Subsystem will get from the source. Others will put data in a predefined location which is then periodically polled by the ingest client software. At Release A, the principal driver in terms of daily electronic L0 data ingest is the data received through the SDPF for the TRMM platform. At Release B, the principal driver of daily electronic L0 data ingest is the data received through EDOS for the AM-1 platform and Landsat-7 data received at EDC. The Moderate Resolution Imaging Spectrometer (MODIS) and Multi-angle Imaging Spectro-Radiometer (MISR) AM-1 instrument data sent to GSFC and LaRC, respectively, comprise the majority of the L0 data to be archived within the Ingest Subsystem for the first several years of ECS operations. The reliable ingest and archiving of this data, and the associated management of Ingest Subsystem resources are the principal drivers in the Ingest Subsystem design. The effects of the interfaces driving the ingest HWCI design on the sizing factors identified in Table 5.1-1 are summarized in Table 5.1-4. Identified impacts on sizing factors and Ingest Subsystem components are estimates of the relative affects that a particular interface to be supported has on the subsystem. Items designated as having a moderate or high impact will drive the capabilities of the individual components as the site supporting that interface.

The Ingest servers and workstation(s) will be directly connected to the DAAC FDDI network, as is illustrated in Figure 5.1-2. The Ingest servers will contain dual-attached station (DAS) cards, which will be dual-homed to separate FDDI concentrators. This provides redundancy so that full connectivity will exist to the servers even in the event of a concentrator failure. The workstation(s) will contain single-attached station (SAS) cards and each will be connected to a single concentrator, but they will also be split across concentrators so that they are not all connected to the same unit. The FDDI concentrators are in turn connected to the FDDI switch. (Refer to section 5.2 of Volume 0 for a general description of DAAC networks.)

The Ingest subsystem will have direct interfaces to the L0 data provider (e.g., NOLAN at LaRC and MSFC, EBnet as GSFC). The L0 router(s) will contain a DAS card which will be connected to the two separate FDDI concentrators, which will in turn be connected to the FDDI switch via a physically cabled FDDI ring.

Table 5.1-4. Ingest HWCI Interface Drivers

External I/F*	Principal supported functions	Affected sizing factor (high or moderate impact)	Impacted Ingest Subsystem components (high or moderate impact)
EDOS	L0 data ingest	I/O, CPU	Client hosts, working storage, L0 archive repository
SDPF	L0 data ingest	I/O, CPU	
NESDIS (NOAA)	Ancillary data ingest	CPU	Client hosts
Landsat-7	L0R data ingest	I/O, CPU	Client hosts, working storage
SCF	TBD		
ADCs	Ancillary data ingest	CPU	Client hosts
Other DAACs	L0 data ingest	CPU	Client hosts, ingest peripherals
IPs	L0 data ingest	CPU	Client hosts
Users	TBD		

* Not all external interfaces are supported or required at Release A

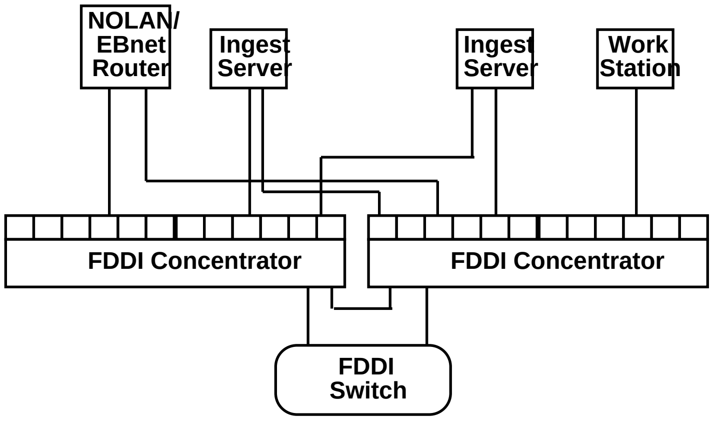


Figure 5.1-2. Ingest Network Connectivity

5.1.2.2 Ingest HWCI Component Description

The Ingest HWCI consists of the ingest client hosts, working storage, and L0 archive repository, and is described in the following paragraphs. Estimates for the number of client hosts and other Ingest Subsystem hardware required to support required ingest functions at each DAAC in the Release A time frame are summarized in Appendix A. Table 5.1-5 summarizes the classes of components which comprise the Ingest HWCI at Release A.

Table 5.1-5. Ingest HWCI Component Descriptions

Component Name	Class/Type	Comments
Client Host	SMP Server W/S / Server W/S	Certain sites require single or multi-processor workstation-based servers, while others require mid-level SMP servers (Release B). See Appendix A for the hardware to client host class mapping at each site.
Working Storage	RAID disk	All sites will utilize one or more RAID units. MSFC and LaRC will utilize working storage RAID units in the Release-A time frame in place of other data repository hardware.
L0 Archive Repository*	Archive Robotics*	An Automated Tape Library (ATL) or similar robotics unit provides the 1 year storage of L0 data. Required at certain sites in Release B and beyond.
	Linear Magnetic Drives*	A typical ATL will contain three media drives (e.g., D3 or 3590 tape) per unit. The three media drives perform input, output, and backup functions.

* Required at Release B at certain sites

Client Hosts

The client hosts will perform pre-processing of the ingested data sufficient both to ensure the basic quality of the received data and to prepare it for archiving and/or further processing. Checks specific to the transfer mechanism (e.g., list of contents, checksums) will be performed, and successful receipt and storage of the data will be acknowledged to the sender and recorded in the ingest logs. It is assumed that for the Release A configuration all data sets will be in a standard format and that metadata will be read either from the file headers, well defined byte locations within the file, or from separate files provided by the data producers rather than by derivation from the data itself. Metadata to be read and logged or generated upon ingest includes data set name, time of ingest, observation time, and granule id. Limited format conversion is required for Release A, but future requirements may include the need for diverse and complex conversions to be performed on data as it is ingested. Additional CPU capabilities necessary to support these conversions will be determined as the required conversions are identified.

Ingest host I/O capabilities must include support for multiple high-bandwidth internal network and peripheral connections. The required capabilities will include technologies such as HiPPI, SCSI II fast/wide, and FDDI for internal network and peripheral connections based on the needs at each site. External network connections may utilize switched technology such as Fiberchannel or ATM for the sites requiring higher bandwidth in the Release B timeframe. Release A network loads are easily accommodated on the dual FDDI rings planned to be implemented at each Release A DAAC, but scaleability of the servers to accommodate additional technologies is an important consideration.

The types of high performance network connections and data rates to be supported by the client host hardware requires significant CPU capabilities. Candidates for client host hardware are discussed in the DAAC-specific configuration appendix and are based on the ingest I/O rate and volume requirements at each site. In general, the client host CPUs are low to mid-level Symmetric Multi-Processing (SMP) 32-bit machines, capable of supporting multiple network (FDDI) and direct-connect (SCSI II) devices. Commonality of components between subsystems is an SDPS design goal. Therefore, the choice of the platform for the ingest client hosts is one that has been made together with that for the data server specifically, as well as processing and other SDPS subsystems, as appropriate. It is typical that workstations from a given manufacturer can be expanded in their CPU and I/O capabilities within a given platform and without significant changes to the system or application software. CPU and I/O capabilities vary at different sites, but should be supported by the same basic platform which may be upgraded with additional capabilities as increased I/O and processing loads demand.

Working Storage

Short term working storage provides a staging area for data moving both into the Ingest Subsystem from network connections or ingest peripherals, and out of the Ingest Subsystem to the Processing and Data Server Subsystems. Short term working storage performance and capacity is driven by the requirement that the Ingest Subsystem reliably and efficiently capture large volumes of L0 data as received from EDOS, SDPF, and other sources. The implementation is also driven by the need for high RMA, as L0 data received from these sources must not be lost or delayed in its availability to other ECS subsystems. This function will be implemented using high performance RAID magnetic disks. Arrays will be shared across ingest hosts and will be sized to handle the anticipated ingest volume, the staging of data out of the Ingest Subsystem, and working storage volume necessary for reformatting, metadata validation, and metadata extraction operations.

Other than the higher RMA requirement, the Ingest Subsystem requirements for working storage are largely the same as those for the Data Server Subsystem and can be satisfied with the same type of hardware. Availability will be 0.999 or better for the ingest of L0 data using a strategy encompassing the use of hot and/or warm spares, the reallocation of working storage space based on data priority within the ingest, processing, and Data Server Subsystems, and the regular and timely transfer of data to long term storage. Working storage devices are RAID 5 units with a minimum of 2.0 days worth of space allocated to ingest working storage. These devices will be sized to the data requirements of the individual sites and will support the functions of acquiring, processing, validating, and archiving L0 data. This volume of working storage allows for one days worth of L0 data to be staged for processing, an additional days worth available for subsequent ingest, and an additional 25% available to service additional Ingest Subsystem needs (e.g., L0 archive retrieval support, pre-processing, quality checking). This equates to a minimum of 625 MB at LaRC and 660 at MSFC. The planned 100 GB units to be shared between working storage and L0 rolling store functions provide more than sufficient capacity at each site. Additional magnetic disk resources are also supplied within the Ingest Subsystem to support items such as client host operating system and application software and L0 archive database directory information.

L0 Archive Repository

The Ingest Subsystem, through a combination of short term working storage and long term storage resources, must pass ingested data to other ECS subsystems as required for processing, long term storage, or other needs of the data system. An additional function of the long term storage portion of the Ingest Subsystem is to store all ingested spacecraft Level 0 data as received from EDOS for a period of one year. This is accomplished through the use of a data repository which is dedicated to the Ingest Subsystem. For Release A, additional RAID devices are planned for use for the L0 archive.

The volumes to be stored at Release B are significant enough to require the addition of a robotics-based archive unit as described below. The L0 archive receives data from the short term working storage and stores it in an automated fashion on high density media. The likely access pattern to this data is one of initial sequential ingest, possible reading of some or all of the ingested data within 24 hours or less, and then infrequent subsequent access. This access pattern, along with the data volumes and ingest rates to be supported at most of the Release B sites, lends itself to a helical scan streaming tape technology (e.g., D3) or linear tape (e.g., 3590) in an automated tape library (ATL) configuration. Multiple tape drives would be available for simultaneous read, write, and hot spare capabilities. The configuration of the ingest L0 archive hardware (e.g., drives, media, robotics, interfaces) will be the same as that for the Data Server Subsystem repositories to support the goal of commonality in development, operations, and maintenance of ECS hardware and software. The possible exception to this would be the incorporation of additional media drives and interfaces to support the increased RMA requirements of the Ingest L0 archive, but this would not significantly change the implementation of the repository. The repositories at each site will be sized to accommodate sufficient media to store one years worth of L0 data, plus a scaling factor of approximately 5% to account for file storage management overhead and data storage inefficiencies. Table 5.1-6 shows the required annual volume of L0 data to be stored at each site.

Table 5.1-6. Annual L0 Storage Volumes

Level 0 (GB/ year)	9/30/97	9/30/98	12/31/99	10/31/01	12/31/02	12/31/03
ASF						
EDC*						
GSFC	0	24440	24663	54732	54907	55301
JPL	0	0	26	26	26	26
LaRC	91	15162	15165	15421	28190	28192
MSFC	96	96	96	99	99	99
NSIDC						
ORNL						
Sum (GB/ year):	187	39698	39950	70278	83222	83618

*Landsat-7 L0R data is stored within the data server repository at EDC.

5.1.3 Failover and Recovery Strategy

The Ingest Subsystem failure and recovery strategy is implemented to satisfy the RMA requirements associated with science data ingest, which require an operational availability of 0.999 and switchover from a primary to backup capability within 15 minutes. The failover/recovery strategy for the Ingest Subsystem is implemented in several ways:

- Recovery from the failure of individual data transfers is initiated automatically, with a tunable number of automated retries before a failure alert is sent to an operator.
- Recovery from a failure in the primary client host CPU involves detection of the failure, resetting the network address of the backup client host to that of the primary, mounting the dual-ported disks shared by the backup and primary, and other minimal software reconfiguration required to make the backup to look identical to the primary.

There are three types of network failures that may affect the Ingest subsystem:

- If the FDDI cable between a host and the FDDI concentrator is severed or damaged, then a new cable would need to be installed. No other configuration would be required.
- If an individual port on the FDDI concentrator fails, then the attached host must be moved to another port, again with no other configuration required.
- Finally, if the entire concentrator fails, then it will have to be replaced, which can be done rapidly since the units require very little configuration.

Note that the above failures result in service interruption only to the workstation(s). For the Ingest servers, there is no single point of failure for the network for receiving L0 data. Since all Ingest servers are attached to two hubs, they will communicate as normal in the event of a cable or concentrator fault, and the applications will be unaware of and unaffected by the event (e.g., L0 ingest will not be interrupted).

Failures in working storage are accommodated in two ways:

- Redundancy is an inherent design feature of RAID devices, typically allowing the failure of one drive in the unit without any loss of data and minimal performance degradation. Many RAID units also incorporate software that keeps track of soft errors as well as hard failures, allowing drives that may be gradually failing to be replaced prior to a hard failure. Even in the event of a hard failure, the unit can continue to operate while a new drive is installed and rebuilt using the data from the other drives in the unit.
- A catastrophic failure of the RAID unit would require that ingest operations be temporarily limited to critical data ingest only. Each Release A site has at least two RAID devices, with one designated as working storage and one as the Level 0 rolling store. Open capacity in the rolling store drive could temporarily be used to take over working storage functions.

Appendix A. Requirements Trace

The Interim Release 1 (Ir1) and TRMM Development (Release A) Level 4 requirements listed in the following table reflect the state of the RTM database on July 15, 1995. The text provided below is a subset of that in the RTM data base and is included to aid the reader in mapping requirements to object classes, CSCs, and CIs.

Table A-1. Requirements Trace(1 of 11)

L4 Rqmt ID	L4 Requirement Text	Object Class, CSC, or CI
S-INS-00010	The INGST CI shall accept Network Ingest Requests to request automated electronic network ingest of a collection of Data. The collection of Data shall describe one or more Data Granules.	InSession
S-INS-00020	The INGST CI shall check the Network Ingest Request to verify that the date/time prior to which the data will remain available is a valid date/time.	InRequest
S-INS-00030	The INGST CI shall authenticate the provider of a Network Ingest Request as an authorized provider of data to be ingested.	CsGateway
S-INS-00040	The INGST CI shall report status to the provider of a Network Ingest Request and to the Error Log indicating successful or unsuccessful authentication of the provider as authorized to submit the request.	CsGateway
S-INS-00050	The INGST CI shall report the following to the MSS event log services: a. Receipt of a network ingest request; b. Response to a network ingest request.	InSession
S-INS-00060	The INGST CI shall report status to the provider of a Network Ingest Request for the following: a. File transfer failure b. File size discrepancies c. Invalid Data Type Identifier d. Missing required metadata e. Metadata parameters out of range f. Data conversion failure g. Failure to archive data h. Inability to transfer data within the specified time window i. Missing required request information j. Successful archive of the data	InSession
S-INS-00070	The INGST CI shall provide the capability to periodically check a location accessible to the ESN for the presence of a Delivery Record file describing data to be ingested. The Delivery Record file shall contain the same information as a Network Ingest Request.	InPollingIngestSession
S-INS-00080	The INGST CI shall read a Delivery Record file describing data to be ingested at a location accessible to the ESN and submit a corresponding Network Ingest Request to be processed.	InDAN

Table A-1. Requirements Trace(2 of 11)

L4 Rqmt ID	L4 Requirement Text	Object Class, CSC, or CI
S-INS-00085	The INGST CI shall report status to the provider of a polling ingest request (delivery record file) for the following: a. File transfer failure; b. File size descrepancies; c. Invalid data type identifier; d. Missing required metadata; e. Metadata parameters out of range; f. Failure to archive data; g. Missing required request information; h. Successful archive of the data.	InPollingIngestSession
S-INS-00090	The INGST CI shall provide the capability for authorized operations staff to set the period between checking for the presence of Delivery Record files.	InThresholdController
S-INS-00100	The INGST CI shall provide the capability to periodically check a location accessible to the ESN for the presence of data granule files.	InPollingIngestSession
S-INS-00110	The INGST CI shall submit an Polling Ingest Request after detecting the presence of data granule files in a location accessible to the ESN. The request shall contain the file location.	InPollingIngestSession
S-INS-00120	The INGST CI shall provide the capability for authorized operations staff to set the period between checking for the presence of external data granule files.	InThresholdController
S-INS-00130	The INGST CI shall interactively accept Hard Media Ingest Requests from operations staff for data to be ingested from hard media.	InMediaIngest
S-INS-00140	The INGST CI shall check the Hard Media Ingest Request to verify that the Media Type is a type supported by the facility to which the request was submitted.	InMediaIngest
S-INS-00150	The INGST CI shall verify that the External Data Provider specified in a Hard Media Ingest Request is an authorized provider of hard media to be ingested.	InMediaIngest
S-INS-00160	The INGST CI shall authenticate that the Hard Media Ingest Request is input by operations staff authorized to ingest hard media data.	InMediaIngest
S-INS-00165	The INGST CI shall read a Delivery Record file describing data to be ingested to determine the files to be ingested after hard media data transfer.	InDAN
S-INS-00170	The INGST CI shall report Hard Media Ingest Request status to the submitting operations staff for the following:	InMediaIngest
S-INS-00180	The INGST CI shall interactively accept Network Ingest Requests from authorized science users for electronic network ingest of a collection of Data from a location accessible via the ESN. The collection of Data shall describe one or more Data Granules.	InNetworkIngest
S-INS-00190	The INGST CI shall check the Network Ingest Request to verify that the date/time prior to which the data will remain available is a valid date/time in a Network Ingest Request entered interactively by a science user.	InRequest
S-INS-00200	The INGST CI shall allow a science user to specify the list of granule files in an interactive Network Ingest Request based on a displayed list of existing files stored on magnetic disk.	InNetworkIngest

Table A-1. Requirements Trace(3 of 11)

S-INS-00205	The INGST CI shall determine the External Data Provider for a Network Ingest Request entered interactively by a science user.	InNetworkIngest
S-INS-00207	The INGST CI shall automatically determine the data volume for each file in the list of granule files for an interactively entered Network Ingest Request.	InNetworkIngest
S-INS-00208	The INGST CI shall authenticate that the interactive science user entering a Network Ingest Request is authorized to request ingest of data.	InNetworkIngest
S-INS-00209	The INGST CI shall report to the Error Log an unauthorized attempt to interactively request ingest of data.	InNetworkIngest
S-INS-00210	The INGST CI shall allow authorized science users to save the contents of an interactively entered Network Ingest Request in a file with a specified file name.	InNetworkIngest
S-INS-00220	The INGST CI shall report status to the interactive submitter of a Network Ingest Request for the following:	InNetworkIngest
S-INS-00221	The INGST CI shall interactively accept Document Ingest Requests from authorized science users for ingest of a single collection of document Data from a location accessible via the ESN. The collection of document Data shall describe one or more document Data Granules.	InNetworkIngest
S-INS-00222	The INGST CI shall check the Document Ingest Request to verify that the date/time prior to which the data will remain available is a valid date/time in a Document Ingest Request entered interactively by a science user.	InRequest
S-INS-00224	The INGST CI shall allow a science user to specify the list of document granule files in an interactive Document Ingest Request based on a displayed list of existing files stored on magnetic disk.	InNetworkIngest
S-INS-00225	The INGST CI shall determine the data provider and assign the Priority Information for a Document Ingest Request entered interactively by a science user.	InNetworkIngest
S-INS-00226	The INGST CI shall automatically determine the data volume for each file in the list of document granule files for an interactively entered Document Ingest Request.	InNetworkIngest
S-INS-00227	The INGST CI shall authenticate that the interactive science user entering a Document Ingest Request is authorized to request ingest of data.	InNetworkIngest
S-INS-00228	The INGST CI shall report to the Error Log an unauthorized attempt to interactively request ingest of document data.	InNetworkIngest
S-INS-00229	The INGST CI shall allow authorized science users to save the contents of an interactively entered Document Ingest Request in a file with a specified file name.	InNetworkIngest
S-INS-00230	The INGST CI shall report status to the interactive submitter of a Document Ingest Request for the following:	InNetworkIngest
S-INS-00235	The INGST CI shall accept ingest Status Requests from science users to determine the status of:	InStatusMonitor
S-INS-00240	The INGST CI shall determine the User Identifier for a science user submitting an ingest Status Request.	InStatusMonitor

Table A-1. Requirements Trace(4 of 11)

S-INS-00250	The INGST CI shall update status on a science user's ongoing Network Ingest Requests, based on User Identifier, to the user.	InStatusMonitor
S-INS-00260	The INGST CI shall provide science users the capability to display the status of the user's ongoing request processing. Displayed status shall include the External Data Provider, ingest Request Identifier, total ingest data volume, and Request State.	InStatusMonitor
S-INS-00270	The INGST CI shall accept ingest Status Requests from authorized operations staff to determine the status of:	InStatusMonitor
S-INS-00280	The INGST CI shall determine the User Identifier for an operations staff member submitting an ingest Status Request.	InStatusMonitor
S-INS-00290	The INGST CI shall authenticate the User Identifier of operations staff requesting status on all ongoing Ingest Requests.	InStatusMonitor
S-INS-00295	The INGST CI shall return an error status to the requester and log information in the Error Log if status is requested on ongoing Ingest Requests from an unauthorized requester.	InStatusMonitor
S-INS-00300	The INGST CI shall return status on ongoing Ingest Requests to an authorized operations staff member.	InStatusMonitor
S-INS-00310	The INGST CI shall provide authorized operations staff the capability to view the status of ongoing ingest processing. Displayed status shall include the External Data Provider, ingest Request Identifier, total ingest data volume, and Request State.	InStatusMonitor
S-INS-00315	The INGST CI shall provide the capability for authorized operations staff to select status of ongoing Ingest Request processing for viewing by means of the External Data Provider.	InStatusMonitor
S-INS-00316	The INGST CI shall accept an Ingest Request from authorized applications.	InRequestManager
S-INS-00317	The INGST CI shall authenticate the User Identifier of an application submitting an Ingest Request.	InRequestManager
S-INS-00318	The INGST CI shall determine the Priority Information for each Ingest Request based on the External Data Provider and the requested ingest priority for the request.	InRequestManager
S-INS-00319	The INGST CI shall add a submitted Ingest Request to a list of Ingest Requests sorted by Priority Information.	InRequestManager
S-INS-00320	The INGST CI shall select an Ingest Request for processing based on the priorities of current requests so long as the number of requests concurrently processed is less than a threshold specified by operations staff. Requests of equal priority will be processed first-in, first-out.	InRequestManager
S-INS-00325	The INGST CI shall determine the ingest start/stop dates and times for all ingested data.	InRequest
S-INS-00330	The INGST CI shall determine the Data Type Identifier for a set of ingested files, whenever the identifier was not provided in the Ingest Request.	InRequest
S-INS-00340	The INGST CI shall report status on processing of an Ingest Request to the Error Log for the following:	InRequest

Table A-1. Requirements Trace(5 of 11)

S-INS-00350	The INGST CI shall accept an ingest Cancellation Request from authorized operations staff to cancel an ongoing ingest request, specifying the ingest Request Identifier.	InRequestController
S-INS-00360	The INGST CI shall authenticate the User Identifier of operations staff submitting an ingest Cancellation Request.	InRequestController
S-INS-00364	The INGST CI shall accept an ingest Cancellation Request from authorized applications to cancel an ongoing Ingest Request, specifying the Request Identifier.	InRequestManager
S-INS-00369	The INGST CI shall authenticate the User Identifier of an application submitting an ingest Cancellation Request.	InRequestManager
S-INS-00380	The INGST CI shall provide authorized operations staff the capability to set thresholds for:	InRequestController
S-INS-00390	The INGST CI shall authenticate the User Identifier of operations staff requesting to set thresholds for concurrent ingest processing.	InRequestController
S-INS-00392	The INGST CI shall report status on ingest Cancellation Requests to the requesting operations staff and to the Error Log for the following:	InRequest; InRequest-Controller
S-INS-00395	The INGST CI shall report status on ingest threshold setup Requests to the requesting operations staff and to the Error Log for the following:	InRequest; InRequest-Controller
S-INS-00396	The INGST CI shall report status on ingest Cancellation Requests to the requesting application and to the Error Log for the following:	InRequest; InRequest-Manager
S-INS-00400	The INGST CI shall convert ingested data into a form accepted by the SDSRV CI / DDSRV CI, for following data types: a. NMC GRIB data.	InGRIBData
S-INS-00403	The INGST CI shall perform the following metadata conversions: a. PB5 time into ECS standard date / time format; b. Binary integer values into ASCII integer format; c. Binary floating point values into ASCII floating point format.	InMetadata
S-INS-00404	The INGST CI shall extract metadata from ingested data into a form accepted by the Science Data Server / Document Data Server, as needed, for the following categories of data: a. Metadata parameters stored by parameter byte order and parameter byte length; b. Metadata parameters stored in PVL format; c. Metadata parameters stored in HDF format; d. Dataset-specific metadata formats	InMetadata
S-INS-00405	The INGST CI shall append the following ingest-specific metadata to metadata corresponding to ingested data:	InMetadata
S-INS-00406	The INGST CI shall check selected parameters from extracted metadata to verify: a. Metadata parameters stored in a dataset specific format, ... f. That date / time values include a valid month, day of month, hour, minute, and second; g. That date / time values include a year value within a range specific for that date / time value.	InMetadata
S-INS-00408	For each data granule specified in an Ingest Request the INGST CI shall determine by means of an Advertisement the appropriate SDSRV CI/DDSRV CI in which to store the data granule.	InDataServerInsertion-Task

Table A-1. Requirements Trace(6 of 11)

S-INS-00409	The INGST CI shall provide the capability to request storage of a data granule by means of a Data Insert Request to the SDSRV CI/DDSRV CI associated with the type of the data granule.	InDataServerInsertion-Task
S-INS-00410	The INGST CI shall provide the capability to electronically transfer data to be ingested via the ESN into a specified ECS storage location.	InDataTransferTask
S-INS-00415	The INGST CI shall provide an interim capability to electronically transfer data to be ingested via the ESN into a specified ECS storage location for early interface testing purposes.	InRequest
S-INS-00420	The INGST CI shall provide the capability for an external application to transfer data to be ingested into a specified ECS storage location.	InNetworkIngest (Release B capability)
S-INS-00425	The INGST CI shall provide the capability to request transfer of data from an 8mm tape.	InMediaIngest
S-INS-00430	The INGST CI shall provide the capability by means of a Working Storage Allocation Request to the Data Server to allocate storage space for data to be transferred to satisfy an ingest request.	InDataTransferTask
S-INS-00440	The INGST CI shall estimate whether data may complete transfer before the date/time prior to which the data will remain available.	InRequest
S-INS-00450	The INGST CI shall retry transfer of data from the External Data Provider N times before the ingest request is failed, where N is a number specified by operations staff.	InDataTransferTask
S-INS-00455	Operations staff shall contact the network operations staff and External Data Provider operations staff to resolve data transfer problems that are not handled automatically.	N/A (Operational Requirement)
S-INS-00460	The INGST CI shall determine the size of each file transferred to ECS whenever file sizes are specified in the corresponding Ingest Request.	InFile
S-INS-00470	The INGST CI shall compare the size of each file after data transfer to ECS with file sizes specified in the corresponding Ingest Request.	InFile
S-INS-00480	The INGST CI shall verify that all files specified in an Ingest Request are successfully transferred to ECS.	InDataTransferTask
S-INS-00490	The INGST CI shall log the following information in an Ingest History Log for each received Ingest Request:	InRequest
S-INS-00500	The INGST CI shall provide operations staff the capability to view selected entries from the Ingest History Log.	InLogMonitor
S-INS-00510	The INGST CI shall provide the capability to select Ingest History Log entries for viewing by the following parameters: ... e. Test or operational mode.	InLogMonitor
S-INS-00520	The INGST CI shall ingest data, provided by the SDPF, from the ESN into the LaRC DAAC, using a file transfer protocol.	InSession
S-INS-00530	The INGST CI shall ingest data, provided by the SDPF, from physical media into the LaRC DAAC as a backup transfer mechanism.	InMediaIngest

Table A-1. Requirements Trace(7 of 11)

S-INS-00540	The INGST CI shall ingest data, provided by the SDPF, from the ESN into the MSFC DAAC using a file transfer protocol.	InSession
S-INS-00550	The INGST CI shall ingest data, provided by the SDPF, from physical media into the MSFC DAAC as a backup transfer mechanism.	InMediaIngest
S-INS-00560	The INGST CI shall ingest Data, provided by the TSDIS, from the ESN into the GSFC DAAC using a file transfer protocol.	InSession
S-INS-00570	The INGST CI shall ingest Data, provided by the TSDIS, from the ESN into the MSFC DAAC using a file transfer protocol.	InSession
S-INS-00580	The INGST CI shall ingest Data, provided by the EDOS, from the ESN into the GSFC DAAC using a file transfer protocol.	InPollingIngestSession
S-INS-00590	The INGST CI shall ingest Data, provided by the EDOS, from the ESN into the LaRC DAAC using a file transfer protocol.	InPollingIngestSession
S-INS-00620	The INGST CI shall ingest data, provided by the DAO, from the ESN into the LaRC DAAC using a file transfer protocol.	InPollingIngestSession
S-INS-00630	The INGST CI shall ingest data, provided by NESDIS, from the ESN into the LaRC DAAC using a file transfer protocol.	InPollingIngestSession
S-INS-00640	The INGST CI shall ingest data, provided by the DAO, from the ESN into the GSFC DAAC using a file transfer protocol.	InPollingIngestSession
S-INS-00670	The INGST CI shall ingest Data, provided by an SCF, from the ESN into the MSFC DAAC using a file transfer protocol.	InNetworkIngest
S-INS-00680	The INGST CI shall ingest Data, provided by an SCF, from the ESN into the LaRC DAAC using a file transfer protocol.	InNetworkIngest
S-INS-00720	The INGST CI shall ingest data, provided by the EOC, from the ESN using a file transfer protocol.	InSession
S-INS-00780	The INGST CI shall ingest data, provided by the Landsat 7 Processing Facility (LPS), from the ESN into the EDC DAAC using a file transfer protocol.	InSession
S-INS-00800	The INGST CI shall ingest Data, provided by Version 0, from the LaRC DAAC using a file transfer protocol.	InSession; InNetworkIngest
S-INS-00810	The INGEST shall ingest Data, provided by Version 0, from the GSFC DAAC on 8mm tape.	InMedia
S-INS-00830	The INGEST shall ingest Data, provided by Version 0, from the MSFC DAAC on 8mm tape.	InMedia
S-INS-00870	The ICLHW CI at the GSFC DAAC shall be capable of ingesting data .for EDOS/ECOM interface testing.	ICLHW
S-INS-00880	The ICLHW CI at the LaRC DAAC shall be capable of ingesting data for EDOS/ECOM interface testing.	ICLHW
S-INS-00990	The ICLHW CI at the LaRC DAAC shall be capable of ingesting data from the SPDF at the nominal daily rate specified in Table E-3 of Appendix E.	ICLHW
S-INS-01000	The ICLHW CI at the LaRC DAAC shall be capable of ingesting data from the SPDF at a maximum daily rate that is three times the nominal rate specified in Table E-3 of Appendix E.	ICLHW

Table A-1. Requirements Trace(8 of 11)

S-INS-01030	The ICLHW CI at the LaRC DAAC shall be capable of ingesting data by network data transfer from the NESDIS, at the nominal daily rate specified in Table E-3 of Appendix E.	ICLHW
S-INS-01040	The INGST CI at the LaRC DAAC shall be capable of receiving data from the SDPF once per day within 24 hours of the last acquisition Client Session.	ICLHW
S-INS-01050	The ICLHW CI at the MSFC DAAC shall be capable of ingesting data from the SPDF at the nominal daily rate specified in Table E-3 of Appendix E.	ICLHW
S-INS-01060	The ICLHW CI at the MSFC DAAC shall be capable of ingesting data from the SPDF at a maximum daily rate that is three times the nominal rate specified in Table E-3 of Appendix E.	ICLHW
S-INS-01070	The ICLHW CI at the MSFC DAAC shall be capable of ingesting data from the TSDIS at the nominal daily rate specified in Table E-3 of Appendix E.	ICLHW
S-INS-01080	The ICLHW CI at the MSFC DAAC shall be capable of ingesting data from the TSDIS at a maximum daily rate that is three times the nominal rate specified in Table E-3 of Appendix E.	ICLHW
S-INS-01100	The INGST CI at the MSFC DAAC shall be capable of receiving data set from the SDPF once per day within 24 hours of the last acquisition Client Session.	ICLHW
S-INS-01136	The ICLHW CI at the GSFC DAAC shall be capable of ingesting data from the DAO at the nominal daily rate specified in Table E-3 of Appendix E.	ICLHW
S-INS-01138	The ICLHW CI at the LaRC DAAC shall be capable of ingesting data from the DAO at the nominal daily rate specified in Table E-3 of Appendix E.	ICLHW
S-INS-01142	The ICLHW CI at the LaRC DAAC shall be capable of ingesting data from NESDIS at the nominal daily rate specified in Table E-3 of Appendix E.	ICLHW
S-INS-60110	The ICLHW CI shall support the hardware resource requirements of the INGST CI and its interface requirements with the operations staff.	ICLHW
S-INS-60120	The ICLHW CI shall have provisions for degraded modes to meet RMA requirements.	ICLHW
S-INS-60140	The ICLHW CI shall have a fail-soft capability to meet RMA requirements.	ICLHW
S-INS-60150	The ICLHW CI shall have provision for Initialization, Recovery, and an orderly shutdown.	ICLHW
S-INS-60160	Startup and initialization of the ICLHW CI shall be completed within 30 minutes (TBR).	ICLHW
S-INS-60170	Shutdown of the ICLHW CI shall be completed within 30 minutes (TBR).	ICLHW
S-INS-60180	The ICLHW CI shall have provision for a fault detection/ fault isolation capability without interfering with operations.	ICLHW
S-INS-60190	The ICLHW CI shall have a status monitoring capability.	ICLHW
S-INS-60210	The ICLHW CI shall support TBD transactions per day, as specified for each release and corresponding DAAC sites.	ICLHW

Table A-1. Requirements Trace(9 of 11)

S-INS-60310	The ICLHW CI shall be capable of operating in a 24 hour per day, 7 days a week mode.	ICLHW
S-INS-60320	The ICLHW CI shall be configured to support the receipt of science data function's Availability (A0) requirement of .99900 and Mean Down Time (MDT) requirement of 2 hours or less.	ICLHW
S-INS-60330	The ICLHW CI shall be capable of supporting system maintenance without impact to normal operations.	ICLHW
S-INS-60410	The ICLHW CI shall provide maintenance interfaces to support the function of System Maintenance.	ICLHW
S-INS-60420	The ICLHW CI shall provide operations interfaces to support the function of System Maintenance.	ICLHW
S-INS-60430	The ICLHW CI platforms shall have provision for interfacing with one or more Local Area Networks (LANs).	ICLHW
S-INS-60510	The electrical power requirements for ICLHW CI equipment shall be in accordance with and the ECS Facilities Plan (DID 302/DV2).	ICLHW
S-INS-60540	The air conditioning requirements for ICLHW CI equipment shall be in accordance with the ECS Facilities Plan (DID 302/DV2).	ICLHW
S-INS-60550	The grounding requirements for ICLHW CI equipment shall be in accordance with ECS Facilities Plan (DID 302/DV2).	ICLHW
S-INS-60560	The fire alarm requirements for ICLHW CI equipment shall be in accordance with ECS Facilities Plan (DID 302/DV2).	ICLHW
S-INS-60570	The acoustical requirements for ICLHW CI equipment shall be in accordance with ECS Facilities Plan (DID 302/DV2).	ICLHW
S-INS-60580	The physical interface requirements between ICLHW CI equipment and the facility shall be in accordance with ECS Facilities Plan (DID 302/DV2).	ICLHW
S-INS-60590	The footprint size and the physical layout of ICLHW CI equipment shall be in accordance with the ECS Facilities Plan (DID 302/DV2).	ICLHW
S-INS-60605	The ICLHW CI shall support test activities throughout the development phase.	ICLHW
S-INS-60610	The following testing shall be performed on the ICLHW CI: a. Unit Testing b. Subsystem testing c. Integration & Testing d. End-to-End testing	ICLHW
S-INS-60620	Internal testing shall be performed on the ICLHW CI which includes tests of hardware functions, and integration testing with other SDPS subsystems.	ICLHW
S-INS-60630	Internal testing shall be performed on the ICLHW CI to verify the internal interfaces to the Data Management, Client, Data Server, Planning, and Data Processing subsystems.	ICLHW
S-INS-60640	Each ICLHW CI element shall be capable of supporting end-to-end test and verification activities of the EOS program including during the pre-launch, spacecraft verification, and instrument verification phases.	ICLHW

Table A-1. Requirements Trace(10 of 11)

S-INS-60650	The ICLHW CI shall be capable of being monitored during testing.	ICLHW
S-INS-60710	The ICLHW CI shall contain the storage and interface resources to support the ingest functions for the TRMM mission instruments of CERES and LIS.	ICLHW
S-INS-60720	The ICLHW CI at the GSFC DAAC shall be sized to support TBD bytes/second at the electronic data ingest interface to support the TRMM mission.	ICLHW
S-INS-60725	The ICLHW CI at the LaRC DAAC shall be sized to support TBD bytes/second at the electronic data ingest interface to support the TRMM mission.	ICLHW
S-INS-60730	The ICLHW CI at the MSFC DAAC shall be sized to support TBD bytes/second at the electronic data ingest interface.	ICLHW
S-INS-60735	The ICLHW CI at the GSFC DAAC shall be sized to store and maintain TBD bytes of data for a 1 year period of time.	ICLHW
S-INS-60740	The ICLHW CI at the LaRC DAAC shall be sized to store and maintain TBD bytes of data for a 1 year period of time.	ICLHW
S-INS-60745	The ICLHW CI at the MSFC DAAC shall be sized to store and maintain TBD bytes of data for a 1 year period of time.	ICLHW
S-INS-60750	The ICLHW CI at the GSFC DAAC shall be sized to temporarily store TBD bytes of ingest data to support the TRMM mission.	ICLHW
S-INS-60755	The ICLHW CI at the LaRC DAAC shall be sized to temporarily store TBD bytes of ingest data to support the TRMM mission.	ICLHW
S-INS-60760	The ICLHW CI at the MSFC DAAC shall be sized to temporarily store TBD bytes of ingest data.	ICLHW
S-INS-60765	The ICLHW CI shall have a switchover time from the primary science data receipt capability to a backup capability of 15 minutes or less.	ICLHW
S-INS-60810	The operating system for each UNIX platform in the ICLHW CI shall conform to the POSIX.2 standard.	ICLHW
S-INS-60820	The ICLHW CI POSIX.2 compliant platform shall have the following utilities installed at a minimum: perl, emacs, gzip, tar, imake, prof, gprof, nm.	ICLHW
S-INS-60830	The ICLHW CI POSIX.2 compliant platform shall have the following POSIX.2 user Portability Utilities installed at a minimum: man, vi.	ICLHW
S-INS-60840	The ICLHW CI POSIX.2 compliant platform shall have the following POSIX.2 Software Development Utilities installed at a minimum: make.	ICLHW
S-INS-60850	The ICLHW CI POSIX.2 compliant platform shall have the following POSIX.2 C-Language Development Utilities installed at a minimum: lex, yacc.	ICLHW
S-INS-60860	The ICLHW CI POSIX.2 compliant platform shall have the following Unix shells installed at a minimum: C shell, Bourne shell, Korn shell.	ICLHW
S-INS-60870	The ICLHW CI POSIX.2 compliant platform shall have on-line documentation or printed documentation for each installed tool.	ICLHW

Table A-1. Requirements Trace(11 of 11)

S-INS-60880	The ICLHW CI POSIX.2 compliant platform shall have installed one or more development environment supporting the following languages:	ICLHW
S-INS-60890	Each development environment associated with the POSIX.2 compliant platform in the ICLHW CI shall have the capability to compile and link strictly conformant POSIX-compliant source code.	ICLHW
S-INS-60895	Each development environment associated with the POSIX.2 compliant platform in the ICLHWCI shall have an interactive source level debugger for ECS supported languages.	ICLHW

Appendix B. Program Design Language (PDL)

The Ingest Program Design Language (PDL) for non-trivial operations is included in this appendix. The PDL is sorted by object class and by object class operation. C++ syntax is used to identify object class operations (e.g., InBOMetadata::Preprocess indicates the Preprocess operation of the InBOMetadata object class). The PDL follows the standards set up in the PDL Program Instruction addendum to the Software Development Plan.

InBOMetadata::Preprocess

```
Call InMetadataTool::PGS_MET_INITt o load target MCF into memory
Call InFile::Read to read metadata file into memory
DoWhile(Call InMetadataTool::GetNext=True)
        Call InSourceMCF::GetParInfo to obtain location of value associated with target
        parameter name
        Read value from appropriate location
        If (Data is Binary) Then
        Call ConvertBintoASCII to map binary expression to corresponding ASCII string via Look
Up
        Table
        EndIf
        Call InMetadataTool::PGS_MET_SET to set the value of attribute in target MCF
End DoWhile
```

int InDAN::Check(char *DAAmsgPtr)

```
Open the DAA Error File; the file would contain one of the following code:
        1.  Accepted
        2.  Invalid DAN Sequence number
        3.  Invalid File Count
        4.  Invalid Data Service
        5.  Invalid Aggregate Length
        6.  Invalid Data Type
        7.  Invalid Directory
        8.  Invalid Time Stamp Format
        9.  Invalid Generation Time Format
        10. Invalid File Size Field
        11. Invalid Time/Date Format
Read the DAA Error code from the DAA file
Close the DAA Error File
if DAA Error code is 1-5
  Construct the InShortDAA class
  Call InShortDAA.FillDAA() to package the DAA message using DAA Error code
          and myDANSeqNo
else
  For each entry in myDataTypeList[]
          Assign DAA Error code to DAAStatus[i]
          Assign myDataTypeList[i].DataTypeId to DataType[i]
          Assign myDataTypeList[i].DatDescriptor to DataDescriptor[i]
  End for
  Construct the InLongDAA class
  Call InLongDAA.FillDAA() to package the DAA message using DAAStatus[],
          DataType[],DataDescriptor[], myDataTypeCount, and myDANSeqNo
endif
return
```

InDAN::InDAN(char *DANFile, char *DAAmsgPtr, int status)

```
Get the total byte size of DANFile
Allocate PVL-Buffer with total DANFile byte size
Open DANFile
Read the whole DANFile into PVL-Buffer
Close DANFile
Call ParsePVL() passing the PVL-Buffer address and the DANFile size to
  extract PVL keywords and Keyword-Values from the PVL-Buffer
Call Check() to verify the DAN components and fills the verification
  results in the DAA data message
```

InDAN::InDAN(DANmsg *DANmsgPtr, char *DAAmsgPtr, int DAAlength, int status)

```
Calculate the total bytes of MsgHeader, EDU_Label, DAN_Label and
  assign to HdrLen
Set PVLptr to HdrLen byte after the first byte of DANmsgPtr
Extract the TotalMessageLength from MsgHeader
Calculate the PVLlen by subtracting TotalMessageLength from HdrLen
Call ParsePVL() passing the PVLptr and PVLlen
Call Check() to verify the DAN components and fills the verification
  results in the DAA message
return
```

int InDAN::ExtractKeyword(char *PVL_stmt, char *Keyword, char *Keyword_Value)

```
*** Task complete by parser() ***
Search for 1st alphanumeric position in PVL_Stmt and assign to StartPtr
Set PVL_Stmt to Key_StartPtr
Search for 1st non-alphanumeric position in PVL_Stmt and assign to EndPtr
Extract StartPtr to EndPtr from PVL_Stmt and assign to Keyword
```

```
Call ExtractValue to extract the keyword value
return
```

int InDAN::ExtractPVLStmt(char* PVL_Buffer, char *PVL_stmt, int PVL_Stmt_Len)

```
*** Task complete by parser() ***
Search for 1st alphanumeric position in PVL_Buffer and
  assign to StartPtr
Search for ';' in PVL_Buffer and assign pocation to EndPtr and PVL_Stmt_Len
Extract PVL_Stmt from PVL-Buffer starting StartPtr byte to Stmt_EndPtr -1
return
```

int InDAN::ExtractValue(char *PVL_stmt, char *Keyword_Value)

```
***Task complete by parser()***
Search for 1st alphanumeric position after '=' in PVl_stmt and assign
  position to StartPtr
Search for 1st non-alphanumeric position after StartPtr and assign
  position to EndPtr
Extract StartPtr to EndPtr from PVL_Stmt and assign to Keyword-Value
return
```

int InDAN::ExtractValue(char *PVL_Stmt, char *Keyword_Value)

```
  Task complete by parser()
  return SUCCESS;
```

int InDAN::FillData (char *IngestType, char *ParsedKeywords)

```
Copy DAN information from daninfo data structure to the private
    member data structure.

return
```

int InDAN::ParsePVL(char* PVL_Buffer, int PVL_Len)

```
If PVL_Len is greater than 0 then call parse()
Call FillDAN() to put the extracted PVL statements into the
        InDAN class data memory
return
```

int InDAN::GetDANSeq(void)

```
return DANSeqNum
```

InDataPreprocessTask::Preprocess

```
Call InDataPreprocessTask::InDataPreprocessTask to create a list to contain files to be inserted
into the Data Server Subsystem
Call InDataType::Preprocess to begin preprocessing of specific data type
```

InDataServerInsertionTask::SendInsert

```
Call Advertising to locate appropriate Data Server to insert data
Call DsCIESDTReferenceCollector::DsCIESDTReferenceCollector to set up Data Server session
Call DsCICommand::DsCICommand to instantiate command object and reference advertisement
Call DsCICommand::SetParameters to include list of file types to be inserted
Call DsCIRequest::DsCIRequest to instantiate object and reference associated Command objects
Call DsCIRequest::SubmitRequest to submit request
```

InDataType::Preprocess

```
Call InDataTypeTemplate::GetDInfo to obtain a list of all required files
Do While(Call InDataPreprocessList::GetNext=True)
    Increment File Counter
    Store File Name
    Get file type of File Name
    Store File Type
    Increment appropriate file type counter
End Do While

If  (required file types exist) Then
    Do While (Counter < number of files)
        Call InFileTypeTemplate::GetFTInfo to obtain file type information
        Instantiate appropriate preprocessing specialization
        If(File Type= Metadata)
            Instantiate DsCIDescriptor
            Call DsCIDescriptor:GetMCF to obtain target MCF
        EndIf
    End Do While
    Do While(Counter < number of files)
        Call appropriate preprocess operation
        Call InMetadataTool::PGS_MET_WRITE
        Instantiate a new file
        If(File Type=Metadata) Then
            Call DsCIDescriptor::Validate
            If (Validate Fails) Then
                Log Errors
                Flag Metadata
            EndIf
```

```
                End If
Else
Send Failure status back to InDataPreprocessTask
EndIf
```

InHDFMetadata::Preprocess()

```
Call InMetadataTool::PGS_MET_INIT to read target MCF into memory
Call appropriate HDF I/O tools to read file metadata
Identify HDF metadata object
DoWhile(Call InMetadataTool::GetNext=True)
        Compare target parameter name with file string
        Call InMetadataTool::PGS_MET_SET to set the value of attribute in target MCF
End DoWhile
```

int InLongDAA::FillDAA (int DAAStatus[], char *DataType[], char *DataDescriptor[], int DataTypeCount, int DANSeqNo)

```
If (DANSeqNO is greater than 0)
  Assign DANSeqNo to myLongDAA.DANSequenceNum
  Assign DataTypeCount to myLongDAA.FileGroupCount
  Init MsgLen to LongDAAmsgHeader
  For i = 1 to DataTypeCount
    Assign DAAStatus[i] to myLongDAA.FileGroup[i].Disposition
    Assign DataType[i] to myLongDAA.FileGroup[i].DataType
    Assign DataDescriptor[i] to myLongDAA.FileGroup[i].Descriptor
         Increment MsgLen by the string length of DataType[i]
      and DataDescriptor[i]
  End for
Else
  write invalid DAN Sequence Number into the event log
End if
Assign the least significant 3 bytes of MsgLen to myLongDAA.MsgLength
return
```

int InLongDAA::GetDAA(LongDAAmsg *DAAmsg, int DAAmsgLen)

```
Assign myLongDAA to DAAmsg
Call InMessage::GetMsgLength() and assign to DAAmsgLen
If (DDNmsgLen is <= zero)
  write Empty DAA data message error into the event log
Endif
```

```
return
```

InLongDAA::InLongDAA()

```
Assign the lease significant byte of InCLongDAAType to myLongDAA.MsgType
return
```

int InLongDAA::FillDAA (int DAAStatus[], char *DataType[], char *DataDescriptor[], int DataTypeCount, int DANSeqNo)

```
If (DANSeqNO is greater than 0)
  Assign DANSeqNo to myLongDAA.DANSequenceNum
  Assign DataTypeCount to myLongDAA.FileGroupCount
  Init MsgLen to LongDAAmsgHeader
  For i = 1 to DataTypeCount
    Assign DAAStatus[i] to myLongDAA.FileGroup[i].Disposition
    Assign DataType[i] to myLongDAA.FileGroup[i].DataType
    Assign DataDescriptor[i] to myLongDAA.FileGroup[i].Descriptor
         Increment MsgLen by the string length of DataType[i]
      and DataDescriptor[i]
  End for
Else
  write invalid DAN Sequence Number into the event log
End if
Assign the least significant 3 bytes of MsgLen to myLongDAA.MsgLength
return
```

int InLongDAA::GetDAA(LongDAAmsg *DAAmsg, int DAAmsgLen)

```
Assign myLongDAA to DAAmsg
Call InMessage::GetMsgLength() and assign to DAAmsgLen
If (DDNmsgLen is <= zero)
  write Empty DAA data message error into the event log
Endif
return
```

InLongDAA::InLongDAA()

```
Assign the lease significant byte of InCLongDAAType to myLongDAA.MsgType
return
```

int InLongDDN::FillDDN (int DDNStatus, char *FileDir[], char *FileId[], int FileCount, int DANSeqNo)

```
If (DANSeqNo is greater than 0)
```

```
  Assign DANSeqNo to myLongDDN.DANSeqNo
  Assign FileCount to myLongDDN.FileCount
  Init MsgLen to byte size of LongDDNmsgHeader
  For i = 1 to FileCount
    Assign DDNStatus[i] to myLongDDN.File[i].Disposition
    Assign FileDir[i] to myLongDDN.File[i].Directory
    Assign FileId[i] to myLongDDN.File[i].FileName
         Increment MsgLen by the string length of FileDir[i] and FileId[i]
  End for
Else
  write invalid DAN Sequence Number into the event log
End if
Assign the least significant 3 bytes of MsgLen to myLongDDN.MsgLength
return
```

int InLongDDN::GetDDN(LongDDNmsg *DDNmsg, int DDNmsgLen)

```
Assign myLongDDN to DDNmsg
Call InMessage::GetMsgLength() and assign to DDNmsgLen
If (DDNmsgLen is <= zero)
  write Empty DAA data message error into the event log
Endif
return
```

InLongDDN::InLongDDN()

```
Assign the lease significant byte of InCLongDDNType to myLongDDN.myMsgType
Return
```

int InMessage::GetMsgLength(char *MsgPtr)

```
If (MsgPtr is not NULL)
  Move 2nd to 4th byte from MsgPtr and assign to MsgLen
else
  Assign zero to MsgLen
endif
return MsgLen
```

InMessage::InMessage()

```
Return
```

InPVMetadata::Preprocess()

```
Call InMetadataTool::PGS_MET_INIT to load target MCF into memory
```

```
DoWhile(Call InMetadataTool::GetNext=True)
        Call InSourceMCF::GetParInfo to match target parameter name with source parameter
name
        Call InFile::Read to read metadata file into memory
        DoUntil(Source Parameter=File String)
                Compare Source Parameter with File String
        End Do Until
        Extract value from parameter -value statement
        Call InMetadataTool::PGS_MET_SET to set the value of attribute in target MCF
End DoWhile
```

int InRequest::InRequest (DANmsg *DANmsgPtr)

```
Call InRequest::Check () to parse the PVL contents and check their validity
If (check is successful)
    Fill InRequest attributes
    Call InRequestProcessHeader::InRequestProcessHeader to checkpoint request processing at-
        tributes
    Call InRequestProcessData::InRequestProcessData to checkpoint request data type processing
        attributes
    Call InRequestFileInfo::InRequestFileInfo to checkpoint request file processing attributes
    Call InRequestSummaryHeader::InRequestSummaryHeader to checkpoint request summary
        attributes
    Call InRequestSummaryData::InRequestSummaryData to checkpoint data type summary at-
        tributes
    Return the object pointer (OID) for this object
Else
    Return error status
endif
```

InRequest::InRequest (char *DANFile)

```
Construct InDAN class using pointer to the DAN file
If (InDAN construction indicates success)
    Call InDAN::FillDAN() to read the DAN contents
    Call InRequest::Check () to parse the PVL contents and check their validity
    If (check is successful)
         Fill InRequest attributes
        Call InRequestProcessHeader::InRequestProcessHeader to checkpoint request processing
            attributes
        Call InRequestProcessData::InRequestProcessData to checkpoint request data type pro-
            cessing attributes
        Call InRequestFileInfo::InRequestFileInfo to checkpoint request file processing attributes
        Call InRequestSummaryHeader::InRequestSummaryHeader to checkpoint request sum-
            mary attributes
        Call InRequestSummaryData::InRequestSummaryData to checkpoint data type summary
            attributes
        Return the object pointer (OID) for this object
    Else
        Return error status
    endif
Else
    Return error status
end if
```

int InRequest::ProcessRequest(void)

```
Call InTransferDataTask to transfer files specified in the request to ECS working storage space
If (transfer is successful)
        Call InRequest::ChangeState to update the substate of the request to "data transferred"
        Call InPreprocessingTask to extract metadata, check the metadata, and perform conver-
            sions and reformatting
    If (preprocessing is successful)
        Call InRequest::ChangeState to update the substate of the request to "data preprocessed"
        Call InPreprocessingTask::InsertData to request insert of data into the Data Server
        If (insert request is accepted)
            Call InRequest::ChangeState to update the critical state of the request to "data insert
                submitted"
            Wait for the Data Server to respond (timeout if Data Server does not respond within op-
                erator-tunable threshold)
            If (insert response received)
                Call InRequest::ChangeState to update the critical state of the request to "data insert
                    completed"
            endif
        endif
    endif
endif
Submit a DDN message to InSession
Wait for InSession to respond (timeout if InSession does not respond within operator-tunable
    threshold)
Invoke the destructor to delete the request
```

int InRequest::Cancel(void)

```
Call GetStatus() to get the current state of the request
If (state indicates in "data transfer")
   Call InDataTransferTask::CancelTransfer() to cancel data transferring
else if (state indicates in "data preprocessing")
   Call InDataPreprocessTask::CancelPreprocess() to cancel data preprocessing
else if (state indicates in "data insertion submitted")
```

```
  Call InDataInsertionTask::CancelInsert() to cancel data insertion
endif
return status
```

int InRequest::ChangeState(char *NewState)

```
Set myRequestState to NewState
Checkpoint NewState to InRequestProcessHeader
return
```

InRequestInfo::AddRequest(int DANSeqNum, int RequestId)

```
Check for duplicate DAN
If duplicate DAN not found
  Allocate memory for the new entry for the RequestInfo List
  Fill the new entry with RequestInfo
  Increment the Request count
EndIf
return
```

InRequestInfo::DeleteRequest(int DANSequenceNum)

```
set FOUND to false
for each RequestInfo in the RequestInfo List
  if RequestInfo[i].DANSequenceNum = DANSequenceNum
        set FOUND to true
        exit the loop
  endif
end for
if FOUND is true
  delete i-th RequestInfo entry from the list
  decrement the RequestInfo count
else
  return no match
endif
return
```

InRequestInfo::GetRequestCount(int *RequestCount)

```
Get the total number of requests from the list
```

InRequestInfo::InRequestInfo()

```
Initialize the request list
```

InRequestInfo::InRequestInfo()

Initialize the request list

InRequestInfo::~InRequestInfo()

Delete any dynamically allocated memory

InRequestInfo::ListRequests()

```
For each request in the list
      print DANSeqNum and RequestId
  END for
  Print Total number of requests
  return
```

InRequestInfo::operator==(const InRequestInfo &r)

Defines an element in the RequestInfo list to be equal
   only if the DANSeqNum and RequestId match.

InRequestInfo::SearchRequest(int DANSequenceNum)

```
Set Found to FALSE
For each entry in the RequestInfo List
   if RequestInfo.DANSequenceNUm = DANSequenceNum
            set FOUND to TRUE
      exit loop
   endif
endfor
return
```

int InRequestList::AddRequest(int *RequestID)

```
Add RequestID to list based on priority
Checkpoint RequestList to data base
return
```

int InRequestList::DeleteRequest(int *RequestID)

```
Delete RequestID from list
Checkpoint RequestList to data base
return
```

int InRequestList::SearchRequest(int *RequestID)

```
Find entry with specified RequestID in list
return
```

int InRequestManager_ C::CancelRequest(int* RequestID)

```
Call InRequest::Cancel () to cancel the request at its current state
return
```

DCEObjRefT* InRequestManager_ C::CreateRequest(DANmsg* DANmsgPtr)

```
Invoke the InRequestManager_S::CreateRequest service to create a distributed request object
return
```

DCEObjRefT* InRequestManager_S::CreateRequest(DANmsg* DANmsgPtr)

```
Create an InRequest object
return
```

int InRequestManager_S::InRequestManager()

```
Call InRequestManager_S::RestoreRequestList to determine if checkpointed requests are avail-
    able and to resubmit them as needed
Listen for request creation or cancellation requests
If (a creation request is received)
    Call InExternalDataProviderThreshold::GetIngestPriority to get the priority for the specified
        request
    Create a pthread for the InRequest object
    Call InRequest::InRequest to create a new request
    Return to listening for a request
Elseif (a cancel request is received)
    Call InRequestManager::CancelRequest to cancel the request
endif
return
```

int InRequestManager_S::RestoreRequestList()

```
Call InRequestProcessHeader::SearchTable to determine whether request information is check-
    pointed in an active state ("request created", "data transferred", "data preprocessed", "data in-
    sert submitted", "data insert completed")
For each checkpointed request
    Call InExternalDataProviderThreshold::GetIngestPriority to get the priority for the specified
        request
    Create a pthread for the InRequest object
    Call InRequest::InRequest to create a new request
```

```
endfor
return
```

InServer::InServer()

Initialize the SessionCount to zero.

InServer::StartServer()

```
Setup Ingest RPC Server:
  Create instance of the InServer object class
    (UUID can be automatically created by constructor or passed to
         the constructor)
  Register Object (Place info about the object in the private state)
  Listen and wait for client request
return
```

InSession::InSession()

Initialize the default constructor prototype

InSession::InSession(char *GatewayBH, char *ClientId, int SessionId)

Initialize the alternate constructor

InSession::~InSession()

Delete any dynamically allocated memory

InSession::InitSessServer(char *SessGWBH)

```
Setup Ingest RPC Session Server:
  Create instance of the InSession object class
    (UUID can be automatically created by constructor or passed to
         the constructor)
  Register Object (Place info about the object in the private state)
  Listen and wait for client request(DAN, DDA...)
return
```

InSession::ProcessRequest(DANmsg *DAN, DAAmsg *DAA)

int InSession::ProcessRequest(DANmsg *DAN, DAAmsg *DAA)

```
  Extract DAN sequence #
  Call SearchRequest to search for duplicate request
  If DAN Sequence # NOT found
    Create new Request to verify request contents, to get the request ID and
      DAN sequence # and to add request onto the Ingest Request List
    Package RequestInfo: Request ID, Sequence #
    Add RequestInfo to the Session RequestInfo List
    Package DAA based on the request verification results
  else
    Log Duplicate DAN - Inform OPERATOR
    Package DAA with appropriate response (Duplicate DAN Sequence #)
  endif
  return
```

InSessionInfo::AddSession(char *NewClient, int SessPID)

```
Append Client and Session PID Information to file
Increment number of sessions open
return
```

InSessionInfo::DeleteSession(int SessionID)

```
set FOUND to false
open Ingest Session file
while not found or end-of-file not reached
  read a record
  match the session id field in the record
  If Session ID found
    delete the record from the file
    set FOUND to true
    decrement number of sessions active
    log client/session deletion
  else
    read next record
  endif
endwhile
If not found
  log error - trying to delete non-existant session
endif
return (found)
```

InSessionInfo::ListSessions(void)

```
Open Session List file
for each session in the session file
```

```
  print SessionId and ClientID
end for
close file
return
```

InSessionInfo::SearchSession (char *CID)

```
set FOUND to false
open Ingest Session file
while not found or end-of-file not reached
  read a record
  match the ClientID field in the record
  If Client ID found
    set FOUND to true
  endif
endwhile
close file
return FOUND
```

InSessionInfo::SearchSession (int SessionId)

```
set FOUND to false
open Ingest Session file
while not found or end-of-file not reached
  read a record
  match the session id field in the record
  If Session ID found
    set FOUND to true
  endif
endwhile
close file
return FOUND
```

int InShortDAA::FillDAA(int DAAStatus, int DANSeqNo)

```
If (DANSeqNo is greater than 0)
  Assign DANSeqNo to myShortDAA.DANSequenceNum
  Assign DAAStatus to myShortDAA.Disposition
Else
  Write invalid DAN Sequence Number into the event log
End if
Assign the byte size of ShortDAAmsg structure to ShortDAAmsgLen
Assign the least significant 3 byte of ShortDAAmsgLen
  to myShortDAA.MsgLength
return
```

int InShortDAA::GetDAA(ShortDAAmsg *DAAmsg, int DAAmsgLen)

```
Assign myShortDAA to DAAmsg
Call InMessage::GetMsgLength() and assign to DAAmsgLen
If (DAAmsgLen is <= zero)
  write Empty DAA data message into the event log
Endif
return
```

InShortDAA::InShortDAA()

```
Assign the lease significant byte of InCShortDAAType to myShortDAA.MsgType
Return
```

InShortDDN::FillDDN(int DDNStatus, int DANSeqNo)

```
If (DANSeqNo is greater than 0)
  Assign DANSeqNo to myShortDDN.DANSequenceNum
  Assign DDNStatus to myShortDDN.Disposition
Else
  write invalid DAN Sequence Number into the event log
End if
Assign the byte size of ShortDDNmsg structure to ShortDDNmsgLen
Assign the least significant 3 byte of ShortDDNmsgLen
  to myShortDDN.MsgLength
return
```

InShortDDN::GetDDN(ShortDDNmsg *DDNmsg, int DDNmsgLen)

```
Assign myShortDDN to DDNmsg
Call InMessage::GetMsgLength() and assign to DDNmsgLen
If (DDNmsgLe is <= zero)
  write Empty DAA data message into the event log
Endif
return
```

InShortDDN::InShortDDN()

```
Assign the lease significant byte of InCShortDDNType to myShortDDN.MsgType
Return
```

Acronyms and Abbreviations

ADC	Affiliated Data Center
AM-1	EOS AM Project (morning spacecraft series)
APID	Application Identifier
ASCII	American Standard Code for Information Interchange
ASF	Alaska SAR Facility (DAAC)
ASTER	Advanced Spaceborne Thermal Emission and Reflection Radiometer (formerly ITIR)
ATL	Automated Tape Library
ATM	Asynchronous Transfer Model
AVHRR	Advanced Very High-Resolution Radiometer
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CD-ROM	Compact Disk - Read Only Memory
CERES	Clouds and Earth's Radiant Energy System
CGI	Common Gateway Interface
CI	Configuration Item
CIESIN	Consortium for International Earth Science Information Network
COTS	Commercial-off-the-shelf
CPU	Central Processing Unit
CSC	Computer System Components
CSCI	Computer Software Configuration Item
CSMS	Communications and Systems Management Segment (ECS)
DAA	DAN Acknowledge
DAAC	Distributed Active Archive Center
DAN	Data Availability Notice
DAO	Data Assimilation Office
DBMS	Database Management System
DCE	Distributed Computing Environment (OSF)
DDA	Data Delivery Acknowledgment
DDN	Data Delivery Notice
DID	Data Ingest Distribution
DID	Data Item Description
DIPHW	Distribution and Ingest Peripheral Management HWCI

DPRHW	Data Repository HWCI
DPS	Data Processing Subsystem
DSS	Data Server Subsystem
EBnet	EOSDIS Backbone network
Ecom	EOSDIS communications system
ECS	EOSDIS Core System
EDC	EROS Data Center (DAAC)
EDOS	EOS Data and Operations System
EDR	Environmental Data Record
EGS	EOS Ground System
EOC	EOS Operations Center (ECS)
EOS	Earth Observing System
EOSDIS	Earth Observing System Data and Information System
FDDI	Fiber Distributed Data Interface
FDF	Flight Dynamics Facility
FNL	FinaL Analysis and Forecast System, Global Analysis
FSMS	File Storage Management System
FIP	File Transfer Protocol
GB	Gigabyte
GDAO	GSFC Data Assimilation Office
GPCP	Global Precipitation Climatology Project
GPI	GOES Precipitation Index
GRIB	Gridded Binary
GSFC	Goddard Space Flight Center
GUI	Graphic User Interface
GVTRMM	Ground Verification Tropical Rainfall Measuring Mission
H/K	Housekeeping
HDF	Hierarchical Data Format
HiPPI	High Performance Parallel Interface
HTML	Hyper-Text Markup Language
HTTP	Hypertext Transport Protocol
HWCI	Hardware Configuration Item
I/F	Interface
I/O	Input/Output
IAS	Instrument Activity Specification
ICD	Interface Control Document

ICLHW	Ingest Client HWCI
ID	Identification
IDR	Interim Design Review
IGS	International Ground Station
IP	International Partners
IR-1	Interim Release-1
IRD	Interface Requirements Document
ISSCP	International Satellite Cloud Climatology Project
JPL	Jet Propulsion Laboratory
L0	Level-0
LaRC	Langley Research Center (DAAC)
LIS	Lightning Imaging Sensor
LOM	Logical Object Model
Mb	Mega bit
MB	Mega byte
Mbps	Mega bits per second
MCF	Metadata Configuration File
MISR	Multi-Angle Imaging SpectroRadiometer
MOC	Mission Operations Center
MODIS	Moderate-Resolution Imaging SpectroRadiometer
MRF	Medium Range Forecast
MSFC	Marshall Space Flight Center
MSS	Management Subsystem
MTBF	Mean Time Between Failure
MTTR	Mean Time To Restore
NESDIS	National Environmental Satellite, Data, and Information Service (NOAA)
NMC	National Meteorological Center (NOAA)
NMC	Network Management Center
NOAA	National Oceanic and Atmospheric Administration
NOLAN	Nascom Operational Local Area Network
NSIDC	National Snow and Ice Data Center (DAAC)
OODCE	Object Oriented DCE
ORNL	Oak Ridge National Laboratory (DAAC)
PDL	Program Design Language
PDR	Preliminary Design Review
PDS	Production Data Set

POAM-II	Polar Ozone and Aerosol Experiment
PS	Project Scientist
PVL	Parameter Value Language
QA	Quality Assurance
RAID	Redundant Array of Inexpensive Disks
RID	Review Item Discrepancy
RMA	Reliability, Maintainability, Availability
RPC	Remote Procedure Call
SAGE II	Stratospheric Aerosol and Gas Experiment
SCF	Science Computing Facility
SCSI	Small Computer System Interface
SDPF	Sensor Data Processing Facility (GSFC)
SDPS	Science Data Processing Segment (ECS)
SDR	System Design Review
SDSRV	Science Data Server CSCI
SFDU	Standard Format Data Unit
SMC	System Management Center (ECS)
SMP	Symmetric Multi-Processing
SNMP	Simple Network Management Protocol
SSM/I	Special Sensor for Microwave/Imaging
STMGT	Storage Resource Management
TBD	To Be Determined
TBR	To Be Replaced
HDF	Hierarchical Data Format
TBS	To Be Supplied
TCP/IP	Transmission Control Protocol/Internet Protocol
TDRSS	Tracking and Data Relay Satellite System
TOMS	Total Ozone Mapping Spectrometer
TRMM	Tropical Rainfall Measuring Mission (joint US-Japan)
TSDIS	TRMM Science Data and Information System
UR	Universal Referencev
UUID	Universal Unique Identifier
W/S	Workstation
WKSHW	Working Storage HWCI
WWW	World Wide Web

Glossary

advertisement	A text description that announces the availability of ECS data or services to ECS users.
advertising service	Through the advertising service, users can search and query descriptions of the data and services available in the network. This data is called advertisements. It is prepared by the data and/or service providers.
affiliated data center (ADC)	A facility not funded by NASA that processes, archives, and distributes Earth science data useful for global change research, with which a working agreement has been negotiated by the EOS program. The agreement provides for the establishment of the degree of connectivity and interoperability between EOSDIS and the ADC needed to meet the specific data access requirements involved in a manner consistent and compatible with EOSDIS services. Such data-related services to be provided to EOSDIS by the ADC can vary considerably for each specific case.
ancillary data	Data other than instrument data required to perform an instrument’s data processing. They include orbit data, attitude data, time information, spacecraft engineering data, calibration data, data quality information, and data from other instruments.
application identifier (APID)	The number assigned by spacecraft mission management that represents the on-board application that generated the telemetry data.
application software	Programs designed for specific functions, such as payroll, accounts payable, inventory control, or property management, generally consisting of source code and object code databases, procedures, and documentation
archive tape library	Archive robotics unit
authorized user	see user, authorized
availability	A measure of the degree to which an item is in an operable and committable state at the start of a "mission" (a requirement to perform its function) when the "mission" is called for an unknown (random) time. (Mathematically, operational availability is defined as the mean time between failures divided by the sum of the mean time between failures and the mean down time [before restoration of function].)
baseline	Identification and control of the configuration of software (i.e. selected software work products and their descriptions) at given points in time.
binary file	A data file whose contents are in binary form (i.e., not encoded)
browse data product	Subsets of a larger data set, other than the directory and guide, generated for the purpose of allowing rapid interrogation (i.e., browse) of the larger data set by a potential user. For example, the browse product for an image data set with multiple spectral bands and moderate spatial resolution might be an image in two spectral channels, at a degraded spatial resolution. The form of browse data is generally unique for each type of data set and depends on the nature of the data and the criteria used for data selection within the relevant scientific disciplines.

calibration	The collection of data required to perform calibration of the instrument science data, instrument engineering data, and the spacecraft engineering data. It includes pre-flight calibration measurements, in-flight calibrator measurements, calibration equation coefficients derived from calibration software routines, and ground truth data that are to be used in the data calibration processing routine.
CCSDS recommendations	Recommendations for spacecraft telemetry and telecommand packet format and protocol made by the Consultative Committee for Space Data Systems.
client	A software component that sends or issues service requests to ECS servers or service providers; a requester of service.
client session	see SESSION
commercial off the shelf (COTS)	COTS is a product, such as an item, material, software, component, subsystem, or system, sold or traded to the general public in the course of normal business operations at prices based on established catalog or market prices (see FAR 15.804-3(c) for explanation of terms.
component	The next lower functional subdivision below "subsystem" in the ECS functional hierarchy.
computer software component (CSC)	A distinct part of a computer software configuration item. CSCs may be further decomposed into other CSCs and computer software units.
computer software configuration item (CSCI)	A configuration item comprised of computer software components and computer software units.
configuration	The functional and physical characteristics of hardware, firmware, software or a combination thereof as set forth in technical document and achieved in a product.
configuration item (CI)	An aggregation of hardware, firmware, software or any of its discrete portions, which satisfies an end use function and is designated for configuration management.
Critical Design Review (CDR)	A detailed review of the element/segment-level design, including such details as program design language for key software modules, and element interfaces associated with a release.
DAAC	see Distributed Active Archive Center
DAAC-unique	Functions and capabilities provided by the DAAC beyond those provided by the core system. The functions will be integrated with ECS via APIs for other similar mechanisms. Examples of DAAC-unique functions include visualization, specialized interfaces, and data set-unique functionality.
Data Archive And Distribution System (DADS)	Included in each DAAC and responsible for archiving and distribution of EOS data and information.
data availability acknowledgment	Status return when a data availability notice cannot be satisfied (e.g., due to a validation error or transmission error).
data availability notice	Notice form a client of data available for ingest.
data availability schedule	Data availability schedule is a schedule indicating the times at which specific data sets will be available from remote DADS, EDOS, the international partners, the ADCs, and other data centers for ingestion by the collocated DADS. The schedules are received directly by the PGS.

data center	A facility storing, maintaining, and making available data sets for expected use in ongoing and/or future activities. Data centers provide selection and replication of data and needed documentation and, often, the generation of user tailored data products.
data ingest request	Request to ingest data.
data product	Data products consist of Level 0 data or Level 1 through Level 4 data products obtained by the PGS from the collocated DADS. These represent the primary input to the product generation process. A collection (1 or more) of parameters packaged with associated ancillary and labeling data, uniformly processed and formatted. Typically uniform temporal and spatial resolution. (Often the collection of data distributed by a data center or subsetted by a data center for distribution.) There are two types of data products: a. Standard: A data product produced at a DAAC by a community consensus algorithm. Typically produced for a wide community. May be produced routinely or on-demand. If produced routinely, typically produced over most or all of the available independent variable space. If produced on-demand, produced only on request from users for particular research needs typically over a limited range of independent variable space. b. Special: A data product produced at a science computing facility by a research status algorithm. May migrate to a community consensus algorithm at a later point. If adequate community interest, may be archived and distributed by a DAAC.
data product levels	Raw data--Data in their original packets, as received from the observer, unprocessed by EDOS. • Level 0--Raw instrument data at original resolution, time ordered, with duplicate packets removed. • Level 1A--Reconstructed unprocessed instrument data at full resolution, time referenced, and annotated with ancillary information, including radiometric and geometric calibration coefficients and georeferencing parameters (i.e. platform ephemeris) computed and appended, but not applied to Level 0 data. • Level 1B--Radiometrically corrected and geolocated Level 1A data that have been processed to sensor units. • Level 2--Derived geophysical parameters at the same resolution and location as the Level 1 data. • Level 3--Geophysical parameters that have been spatially and/or temporally re-sampled (i.e., derived from Level 1 or Level 2 data). • Level 4--Model output and/or results of lower level data that are not directly derived by the instruments. Data Levels 1 through 4 as defined in the EOS Data Panel Report. Consistent with the Committee on Data Management and Computation and Earth Science and Applications Data System definitions.
data server	Either the data server subsystem as a whole, or a specific instance of a data server. A data server is a (hardware/software) entity that accepts, stores, and distributes EOS (and other) data, for both other subsystems within ECS and external users.
data server insert request	Request to insert data into a data server.
data set	A logically meaningful grouping or collection of similar or related data.
data type	A particular type of data handled by a particular data server. An example of a data type might be MODIS Level 1a products, etc.

data type taxonomy	A classification of earth science and related data into types.
definitive attitude data	Down-linked attitude data received with Level 0 data.
definitive orbit data	Down-linked orbit (ephemeris) data received with level 0 data.
delivered algorithm packages	The full content of data and information delivered by a data producer during the process of standard product Algorithm Integration & Test, including all elements defined as minimum content within Volume 4 of the Science User's Guide, available at PDR.
Distributed Active Archive Center (DAAC)	An EOSDIS facility which generates, archives, and distributes EOS Standard Products and related information for the duration of the EOS mission. An EOSDIS DAAC is managed by an institution such as a NASA field center or a university, per agreement with NASA. Each DAAC contains functional elements for processing data (the PGS), for archiving and disseminating data (the DADS), and for user services and information management (elements of the IMS). ASF -- Alaska SAR Facility EDC -- EROS Data Center GSFC -- Goddard Space Flight Center JPL -- Jet Propulsion Laboratory LaRC -- Langley Research Center MSFC -- Marshall Space Flight Center NSIDC -- National Snow and Ice Data Center
EDOS data unit (EDU)	The message packet generated by EDOS that contains the reconstructed spacecraft telemetry packet.
engineering data	All data available on-board about health, safety, environment, or status of the spacecraft and instruments. • housekeeping data: The subset of engineering data required for mission and science operations. These include health and safety, ephemeris, and other required environmental parameters. • instrument engineering data: All non-science data provided by the instrument. • platform engineering data: The subset of engineering data from platform sensor measurements and on-board computations. • spacecraft engineering data: The subset of engineering data from spacecraft sensor measurements and on-board computations.
EOS Data and Operations System (EDOS) production data set	Data sets generated by EDOS using raw instrument or spacecraft packets with space-to-ground transmission artifacts removed, in time order, with duplicate data removed, and with quality/ accounting (Q/A) metadata appended. Time span, or number of packets, encompassed in a single data set are specified by the recipient of the data. These data sets are equivalent to Level 0 data formatted with Q/A metadata. For EOS, the data sets are composed of: instrument science packets, instrument engineering packets, spacecraft housekeeping packets, or onboard ancillary packets with quality and accounting information from each individual packet and the data set itself and with essential formatting information for unambiguous identification and subsequent processing.
ephemeris data	See "orbit data"
external data provider	An external data source providing data to be ingested in SDPS.
format	Format of data -- ASCII, binary, etc.

granule	The smallest aggregation of data that is independently managed (i.e., described, inventoried, retrievable). Granules may be managed as logical granules and/or physical granules.
granule location	The name of the product where this granule is located.
hardware	That combination of subcontracted, COTS, and government furnished equipment (e.g., cables and computing machines) that are the platforms for software.
hardware configuration item (HWCI)	A configuration item comprised of hardware components.
HDF file	A data file whose format follows the NCSA Hierarchical Data Format standard, as well as ECS-developed extensions thereto.
I/O access	A read or write by a process to a data file.
ingest status request	Request for status on a data ingest request.
insert request	Request to insert data into the archive.
interface classes	The interfaces offered by a class of objects or object collections. User, for example, in the context of Service Classes to denote the collection of interfaces supported by this service class.
interface definition language (IDL)	IDL provides uniform semantics for all interfaces.
interface(s)	The functional and physical characteristics required to exist at a common boundary.
maintainability	The measure of the ability of an item to be retained in or restored to a specified condition when maintenance is performed by personnel having specified skill levels, using prescribed procedures and resources, at each prescribed level of maintenance and repair. (The probability that maintenance, both corrective and preventive, can be performed in a specified amount of time using a specified set of prescribed procedures and resources expressed as MTTR). Maintainability is the function of design.
mean down time (MDT)	Sum of the mean time to repair MTTR, plus the average administrative logistic delay times.
mean time between failure (MTBF)	The reliability result of the reciprocal of a failure rate that predicts the average number of hours that an item, assembly or piece part will operate within specific design parameters. (MTBF=1/(l) failure rate; (l) failure rate = # of failures/operating time.
mean time to repair (MTTR)	The mean time required to perform corrective maintenance to restore a system/equipment to operate within design parameters. It is a basic measure of maintainability: The sum of corrective maintenance times at any specific level of repair, divided by the total number of failures within an item repaired at that level, during a particular interval under stated conditions.
metadata	Information about data sets which is provided to the ECS by the data supplier or the generating algorithm and which provides a description of the content, format, and utility of the data set. Metadata may be used to select data for a particular scientific investigation. It is “data about data” used to facilitate database searches. Types of metadata include: product metadata (data describing a particular product, such as when it was generated, etc.) and algorithm metadata (data describing science software)

object	Identifiable encapsulated entities providing one or more services that clients can request. Objects are created and destroyed as a result of object requests. Objects are identified by client via unique reference.
object implementation	Code and data that realizes target object's behavior.
operations personnel	Same as operations staff.
operations staff	Generic term for personnel who have the responsibility to operate, monitor, and control SDPS. Also can be, one of the DAAC operations staff assigned to the ingest or data server subsystems, i.e., Data Archive Analyst, Data Ingest Technician, Data Distribution Technician, Data Base Administrator, etc.
orbit data	Data that represent spacecraft locations. Orbit (or ephemeris) data include: Geodetic latitude, longitude and height above an adopted reference ellipsoid (or distance from the center of mass of the Earth); a corresponding statement about the accuracy of the position and the corresponding time of the position (including the time system); some accuracy requirements may be hundreds of meters while other may be a few centimeters.
p = v metadata	Label = value where label is a field name and value is either a single value or list of values
Preliminary Design Review (PDR)	PDR is held for each ECS Segment. The PDR addresses the design of the segment-level capabilities and element interfaces through all ECS releases. The PDR also addresses prototyping results and how the results of both Contractor and Government prototyping efforts, studies, and user experience with EOSDIS Version 0 have been incorporated into the ECS design for each respective Segment.
process	An executing program.
quick-look data	Data received during one TDRSS contact period which have been processed to Level 0 (to the extent possible for data from a single contact).
reliability	Reliability is the function of design. It is the probability that system/equipment will operate within design parameters under stated conditions, for a specified interval expressed as MTBF.
report	Documentation of some automated (such as standards checking) or manual (such as evaluation of a science software delivery) activity.
requirement	A statement to which the developed system must comply. Varieties of requirements: Levels 2, 3, 4; performance, functional, design, interface.
requirements traceability	There are three recognized levels of requirements on the ECS Project: • ESDIS (Level 2) • ECS System (Level 3) • ECS Detailed Subsystem (Level 4) Traceability is the verification and validation of the parents and children of ECS Levels 2,3,4 requirements down to release and subsystem levels. Analysis is done by the ECS Project System and Subsystem engineering.

reusable software	Software developed in response to the requirements for one application that can be used, in whole or in part, to satisfy the requirements of another application.
scenario	A description of the operation of the system in user's terminology including a description of the output response for a given set of input stimuli. Scenarios are used to define operations concepts.
science user	A user the SDPS from the scientist community or other user community that originates service requests.
SDP Toolkit	A set of SDPS-standard API between science algorithms and the process execution service for status reporting and process control
server	A software component that receives and executes service requests (e.g., the LIM, the DIM, the data server, the PLANG CI).
service	A grouping of functional requirements as listed in a specification. For example, in the Level 3 requirements, IMS "services" are System Access, Information Search, etc.
session	The logical context assigned to a user or a client in which a set of service requests are performed. Sessions associate and manage the resources and results sets that are allocated and generated as a result of the processing of service requests. A session retains information associated with the execution of service requests so that it is accessible to subsequent service requests. Service requests may utilize resources and results sets allocated and produced by other service requests belonging to the same session. Service requests issued in the context of one session cannot utilize the resources managed by another session. There are two kinds of sessions, client sessions and user sessions. Sessions have the following states: a. Active: The session is established and will allow service requests to allocate and access session resources. b. Suspended: The session is established, but will not accept service requests. Session resources are saved but not accessible. c. Terminated: The processing of service requests in the session's context is no longer possible. Session resources have been returned to the system.
session, client	A client session supports interactions between a client and a server. Client sessions associate and manage the resources and results sets that are allocated and generated by the server.
simulated data	...same as test data
status	Status is information regarding schedules, hardware and software configuration, exception conditions, or processing performance. This information is exchanged with the DADS, and is provided to the system management center (SSMC). The SSMC may also receive information regarding schedule conflicts that have not been resolved with the IMS.
status request	Request for status of archive insert and retrieval requests (also need this for ingest and distribution).
universal reference	A uniform model for referencing objects throughout SDPS which each SDPS service will understand and support.

user	• Any person accessing the EOSDIS. • Authorized users are users who have viable EOSDIS accounts, and who may therefore make EOSDIS data requests. These users may be affiliated or unaffiliated. Affiliated users are those who are sponsored by one of the parties to the Earth Observations-International Coordination Working Group (EAU-ICWG) data policy. Each party is responsible for ensuring that all its affiliated users comply with the EO-ICWG data policy. Use of data by affiliated users is classified in one of three categories, defined in the EO-ICWG data policy: + Research Use: A study or an investigation in which the user affirms (1) the aim is to establish facts or principles; (2) the data will not be sold or reproduced or provided to anyone not covered by this or another valid affirmation; (3) the results of the research will be submitted for publication in the scientific literature; and (4) detailed results of the research will be provided to the sponsoring spacecraft operator as agreed between the researcher and the sponsoring spacecraft operator. In the context of EOSDIS , this means that NASA-affiliated users must make available to the research community their detailed results, including data, algorithms, and models at the time their research is accepted for publication, and that the data
World Wide Web browser	Software (local or remote) that allows a user to Access the WWW either textually or graphically. WWW is a mechanism for connecting Internet via a set of hypertext documents.